PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04J 1/02

A2

(11) International Publication Number: WO 98/19410

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/18911

(22) International Filing Date: 22 October 1997 (22.10.97)

(30) Priority Data:

9622728.5	31 October 1996 (31.10.96)	GB
9720550.4	26 September 1997 (26.09.97)	GB

(71) Applicant: DISCOVISION ASSOCIATES [US/US]; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

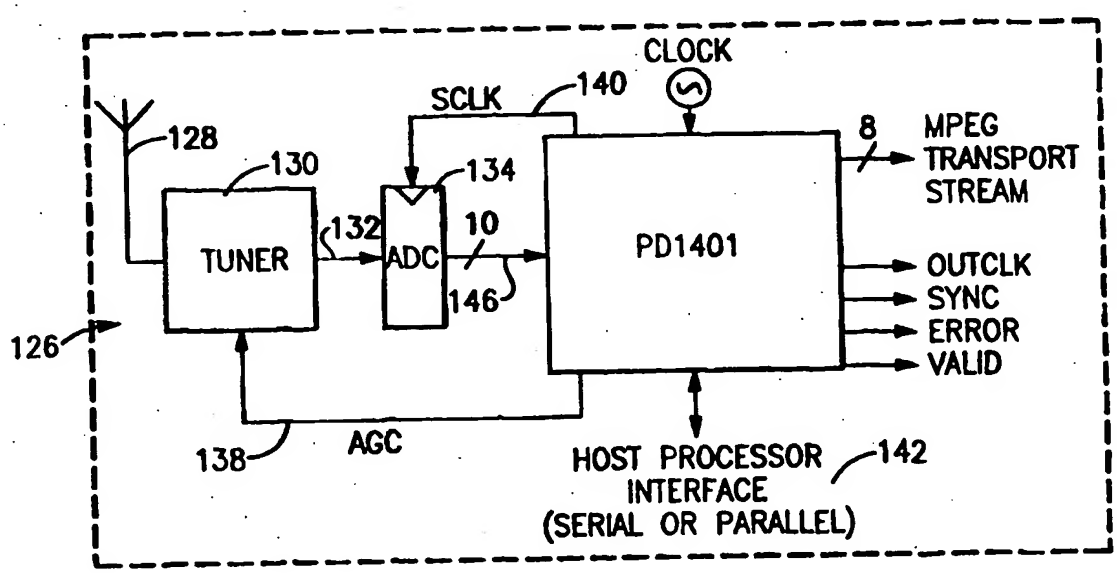
(72) Inventors: ALAM, Dawood; 15 Westbury Park, Durdham Down, Bristol BS6 7JA (GB). COLLINS, Matthew, James; Flat 1, 4 New King Street, Bath BA1 2BN (GB). DAVIES, David, Huw; 6 Glen Brook, Glen Drive, Stoke Bishop, Bristol BS9 1SB (GB). KEEVILL, Peter, Anthony; 7 Junction Road, Oldfield Park, Bath BA2 3NQ (GB). NOLAN, John, Matthew; 19 The Firs, Combe Down, Bath, Somerset BA2 5ED (GB). FOXCROFT, Thomas; 52B Pembroke Road, Clifton, Bristol BS8 3DT (GB). PARKER, Jonathan; 66 Third Avenue, Oldfield Park, Bath BA2 3NZ (GB).

(74) Agent: BICKEL, Arthur, S.; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SINGLE CHIP VLSI IMPLEMENTATION OF A DIGITAL RECEIVER EMPLOYING ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING



(57) Abstract

The invention provides a single chip implementation of a digital receiver for multicarrier signals that are transmitted by orthogonal frequency division multiplexing. Improved channel estimation and correction circuitry are provided. The receiver has highly accurate sampling rate control and frequecy control circuitry. BCH decoding of tps data carriers is achieved with minimal resources with an arrangement that includes a small Galois field multiplier. An improved FFT window synchronization circuit is coupled to the resampling circuit for locating the boundary of the guard interval transmitted with the active frame of the signal. A real-time pipelined FFT processor is operationally associated with the FFT window synchronization circuit and operates with reduced memory requirements.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

SINGLE CHIP VLSI IMPLEMENTATION OF A DIGITAL RECEIVER EMPLOYING ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING

This invention relates to receivers of electromagnetic signals employing multicarrier modulation. More particularly this invention relates to a digital receiver which is implemented on a single VLSI chip for receiving transmissions employing orthogonal frequency division multiplexing, and which is suitable for the reception of digital video broadcasts.

Coded orthogonal frequency division multiplexing ("COFDM") has been proposed for digital audio and digital video broadcasting, both of which require efficient use of limited bandwidth, and a method of transmission which is reliable in the face of several effects. For example the impulse response of a typical channel can be modeled as the sum of a plurality of Dirac pulses having different delays. Each pulse is subject to a multiplication factor, in which the amplitude generally follows a Rayleigh law. Such a pulse train can extend over several microseconds, making unencoded transmission at high bit rates unreliable. In addition to random noise, impulse noise, and fading, other major difficulties in digital terrestrial transmissions at high data rates include multipath propagation, and adjacent channel interference, where the nearby frequencies have highly correlated signal variations. COFDM is particularly suitable for these applications. In practical COFDM arrangements, relatively small amounts of data are modulated onto each of a large number of carriers that are closely spaced in frequency. The duration of a data symbol is increased in the same ratio as the number of carriers or subchannels, so that inter-symbol interference is markedly reduced.

Multiplexing according to COFDM is illustrated in Figs. 1 and 2, wherein the spectrum of a single COFDM carrier or subchannel is indicated by line 2. A set of carrier frequencies is indicated by the superimposed waveforms in Fig. 2, where orthogonality conditions are satisfied. In general two real-valued functions are orthogonal if

$$\int_a^b \psi_p(t)\,\psi_q^*(t)\,dt = K \qquad (1)$$

where K is a constant, and K = 0 if p ≠ q; K ≠ 0 if p = q. Practical encoding and decoding of signals according to COFDM relies heavily on the fast Fourier transform ("FFT"), as can be appreciated from the following equations.

The signal of a carrier c is given by

$$s_c(t) = A_c(t)\,e^{j[\omega_c t + \phi_c(t)]} \qquad (2)$$

where A_c is the data at time t, ω_c is the frequency of the carrier, and ϕ_c is the phase. N carriers in the COFDM signal is given by

$$s_s(t) = (1/N) \sum_{n=0}^{N} A_n(t)e^{j[\omega_n t + \phi_n(t)]} \qquad (3)$$

$$\omega_n = \omega_0 + n\Delta\omega \qquad (4)$$

Sampling over one symbol period, then

$$\phi_c(t) \Rightarrow \phi_n \qquad (5)$$

$$A_c(t) \Rightarrow A_n \qquad (6)$$

With a sampling frequency of 1/T, the resulting signal is represented by

$$s_s(t) = (1/N) \sum_{n=0}^{N} A_n(t)e^{j[(\omega_n + n\Delta\omega)kT + \phi_n]} \qquad (7)$$

Sampling over the period of one data symbol $\tau = NT$, with $\omega_0 = 0$,

$$s_s(kT) = (1/N) \sum_{n=0}^{N-1} A_n e^{j\phi_n} e^{j(n\Delta\omega)kT} \qquad (8)$$

which compares with the general form of the inverse discrete Fourier transform:

$$g(kT) = (1/N) \sum_{n=0}^{N-1} G(n/(kT))e^{j\pi n(k/N)} \qquad (9)$$

In the above equations $A_n e^{j\phi_n}$ is the input signal in the sampled frequency domain, and $s_s(kT)$ is the time domain representation. It is known that increasing the size of the FFT provides longer symbol durations and improves ruggedness of the system as regards echoes which exceed the length of the guard interval. However computational complexity increases according to $N\log_2 N$, and is a practical limitation.

In the presence of intersymbol interference caused by the transmission channel, orthogonality between the signals is not maintained. One approach to this problem has been to deliberately sacrifice some of the emitted energy by preceding each symbol in the time domain by an interval which exceeds the memory of the channel, and any multipath delay. The "guard interval" so chosen is large enough to absorb any intersymbol interference, and is established by preceding each symbol by a replication of a portion of itself. The replication is typically a cyclic extension of the terminal portion

of the symbol. Referring to Fig. 3, a data symbol 4 has an active interval 6 which contains all the data transmitted in the symbol. The terminal portion 8 of the active interval 6 is repeated at the beginning of the symbol as the guard interval 10. The COFDM signal is represented by the solid line 12. It is possible to cyclically repeat the initial portion of the active interval 6 at the end of the symbol.

Transmission of COFDM data can be accomplished according to the known general scheme shown in Fig. 4. A serial data stream 14 is converted to a series of parallel streams 16 in a serial-to-parallel converter 18. Each of the parallel streams 16 is grouped into x bits each to form a complex number, where x determines the signal constellation of its associated parallel stream. After outer coding and interleaving in block 20 pilot carriers are inserted via a signal mapper 22 for use in synchronization and channel estimation in the receiver. The pilot carriers are typically of two types. Continual pilot carriers are transmitted in the same location in each symbol, with the same phase and amplitude. In the receiver, these are utilized for phase noise cancellation, automatic frequency control, and time/sampling synchronization. Scattered pilot carriers are distributed throughout the symbol, and their location typically changes from symbol to symbol. They are primarily useful in channel estimation. Next the complex numbers are modulated at baseband by the inverse fast fourier transform ("IFFT") in block 24. A guard interval is then inserted at block 26. The discrete symbols are then converted to analog, typically low-pass filtered, and then upconverted to radiofrequency in block 28. The signal is then transmitted through a channel 30 and received in a receiver 32. As is well known in the art, the receiver applies an inverse of the transmission process to obtain the transmitted information. In particular an FFT is applied to demodulate the signal.

A modern application of COFDM has been proposed in the European Telecommunications Standard ETS 300 744 (March 1997), which specifies the framing structure, channel coding, and modulation for digital terrestrial television. The specification was designed to accommodate digital terrestrial television within the existing spectrum allocation for analog transmissions, yet provide adequate protection against high levels of co-channel interference and adjacent channel interference. A flexible guard interval is specified, so that the system can support diverse network configurations, while maintaining high spectral efficiency, and sufficient protection against co-channel interference and adjacent channel interference from existing PAL/SECAM services. The noted European Telecommunications Standard defines two modes of operation. A "2K mode", suitable for single transmitter operation and for small single frequency networks with limited transmitter distances. An "8K mode" can be used for either single transmitter operation or for large single frequency networks. Various levels of quadrature amplitude

modulation ("QAM") are supported, as are different inner code rates, in order to balance bit rate against ruggedness. The system is intended to accommodate a transport layer according to the Moving Picture Experts Group ("MPEG"), and is directly compatible with MPEG-2 coded TV signals (ISO/IEC 13818).

In the noted European Telecommunications Standard data carriers in a COFDM frame can be either quadrature phase shift keyed ("QPSK"), 16-QAM, 64-QAM, non-uniform 16-QAM, or non-uniform 64-QAM using Gray mapping.

An important problem in the reception of COFDM transmission is difficulty in maintaining synchronization due to phase noise and jitter which arise from upconversion prior to transmission, downconversion in the receiver, and the front end oscillator in the tuner, which is typically a voltage controlled oscillator. Except for provision of pilot carriers to aid in synchronization during demodulation, these issues are not specifically addressed in the noted European Telecommunications Standard, but are left for the implementer to solve.

Basically phase disturbances are of two types. First, noisy components which disturb neighbor carriers in a multicarrier system are called the "foreign noise contribution" ("FNC"). Second, a noisy component which disturbs its own carrier is referred to as the "own noise contribution".

Referring to Fig. 5, the position of ideal constellation samples are indicated by "x" symbols 34. The effect of foreign noise contribution is stochastic, resulting in Gaussian-like noise. Samples perturbed in this manner are indicated on Fig. 5 as circles 36. The effects of the own noise contribution is a common rotation of all constellation points, indicated as a displacement between each "x" symbol 34 and its associated circle 36. This is referred to as the "common phase error", which notably changes from symbol to symbol, and must therefore be recalculated each symbol period T_S. The common phase error may also be interpreted as a mean phase deviation during the symbol period T_S.

In order for the receiver 32 to process the data symbols in a practical system, a mathematical operation is performed on the complex signal representing each data symbol. Generally this is an FFT. For valid results to be obtained, a particular form of timing synchronization is required in order to align the FFT interval with the received data symbol.

It is therefore a primary object of the invention to provide a highly integrated, low cost apparatus for the reception of digital broadcasts, such as terrestrial digital video broadcasts, which is implemented on a single VLSI chip.

It is another object of the invention to provide an improved method and apparatus for synchronizing a received data symbol with an FFT window in signals transmitted according to COFDM.

It is yet another object of the invention to improve the stability of digital multicarrier receivers in respect of channel estimation.

It is still another object of the invention to improve the automatic frequency control circuitry employed in multicarrier digital receivers.

It is a further object of the invention to improve the automatic sampling rate control circuitry employed in multicarrier digital receivers.

The invention provides a digital receiver for multicarrier signals that are transmitted by orthogonal frequency division multiplexing. The multicarrier signal carries a stream of data symbols having an active interval, and a guard interval in which the guard interval is a replication of a portion of the active interval. In the receiver an analog to digital converter is coupled to a front end amplifier. An I/Q demodulator is provided for recovering in phase and quadrature components from data sampled by the analog to digital converter, and an automatic gain control circuit is coupled to the analog to digital converter. In a low pass filter circuit accepting I and Q data from the I/Q demodulator, the I and Q data are decimated and provided to a resampling circuit. An interpolator in the resampling circuit accepts the decimated I and Q data at a first rate and outputs resampled I and Q data at a second rate. An FFT window synchronization circuit is coupled to the resampling circuit for locating a boundary of the guard interval. A real-time pipelined FFT processor is operationally associated with the FFT window synchronization circuit. Each stage of the FFT processor has a complex coefficient multiplier, and an associated memory with a lookup table defined therein for multiplicands being multiplied in the complex coefficient multiplier. Each multiplicand in the lookup table is unique in value. A monitor circuit responsive to the FFT window synchronization circuit detects a predetermined indication that a boundary between an active symbol and a guard interval has been located.

According to an aspect of the invention the FFT window synchronization circuit has a first delay element accepting currently arriving resampled I and Q data, and outputting delayed resampled I and Q data. A subtracter produces a signal representative of the difference between the currently arriving resampled I and Q data and the delayed resampled I and Q data. In a first circuit the subtracter output signal is converted to a signal having a unipolar magnitude, which is preferably the absolute value of the signal provided by the subtracter. A second delay element stores the output signal of the first circuit, and a third delay element receives the delayed output of the second delay element. In a second circuit a statistical relationship is calculated between data stored in the second delay element and data stored in the third delay element. The output of the FFT window synchronization circuit is representative of the statistical relationship.

Preferably the statistical relationship is the F ratio. The FFT processor is capable of operation in a 2K mode and in an 8K mode.

The FFT processor has an address generator for the memory of each stage, which accepts a signal representing the order dependency of a currently required multiplicand, and generates an address of the memory wherein the currently required multiplicand is stored. In a further aspect of the invention each multiplicand is stored in the lookup table in order of its respective order dependency for multiplication by the complex coefficient multiplier, so that the order dependencies of the multiplicands define an incrementation sequence. The address generator has an accumulator for storing a previous address that was generated thereby, a circuit for calculating an incrementation value of the currently required multiplicand responsive to the incrementation sequence, and an adder for adding the incrementation value to the previous address.

In another aspect of the invention there are a plurality of incrementation sequences. The multiplicands are stored in row order, wherein in a first row a first incrementation sequence is 0, in a second row a second incrementation sequence is 1, in a third row first and second break points B1, B2 of a third incrementation sequence are respectively determined by the relationships

$$B1_{M_N} = 4^N B1_{M_N} - \sum_{n=0}^{N-1} 4^n$$

$$B2_{M_N} = \sum_{n=0}^{N} 4^n$$

and in a fourth row a third break point B3 of a third incrementation sequence is determined by the relationship

$$B3_{M_N} = 2 \times 4^N + 2$$

wherein M_N represents the memory of an Nth stage of the FFT processor.

The receiver provides channel estimation and correction circuitry. Pilot location circuitry receives a transformed digital signal representing a frame from the FFT processor, and identifies the position of pilot carriers therein. The pilot carriers are spaced apart in a carrier spectrum of the transformed digital signal at intervals K and have predetermined magnitudes. The pilot location circuitry has a first circuit for computing an order of carriers in the transformed digital signal, positions of said carriers being calculated modulo K. There are K accumulators coupled to the second circuit for accumulating magnitudes of the carriers in the transformed digital signal, the accumulated magnitudes defining a set. A correlation circuit is provided for correlating

K sets of accumulated magnitude values with the predetermined magnitudes. In the correlation a first member having a position calculated modulo K in of each of the K sets is uniquely offset from a start position of the frame.

According to another aspect of the invention the pilot location circuitry also has a bit reversal circuit for reversing the bit order of the transformed digital signal.

According to yet another aspect of the invention amplitudes are used to represent the magnitudes of the carriers. Preferably the magnitudes of the carriers and the predetermined magnitudes are absolute values.

In a further aspect of the invention the correlation circuitry also has a peak tracking circuit for determining the spacing between a first peak and a second peak of the K sets of accumulated magnitudes, wherein the first peak is the maximum magnitude, and the second peak is the second highest magnitude.

The channel estimation and correction circuitry also has an interpolating filter for estimating the channel response between the pilot carriers, and a multiplication circuit for multiplying data carriers output by the FFT processor with a correction coefficient produced by the interpolating filter.

The channel estimation and correction circuitry also has a phase extraction circuit accepting a data stream of phase-uncorrected I and Q data from the FFT processor, and producing a signal representative of the phase angle of the uncorrected data. The phase extraction circuit includes an accumulator for the phase angles of succeeding phase-uncorrected I and Q data.

According to an aspect of the invention the channel estimation and correction circuitry includes an automatic frequency control circuit coupled to the phase extraction circuit, in which a memory stores the accumulated common phase error of a first symbol carried in the phase-uncorrected I and Q data. An accumulator is coupled to the memory and accumulates differences between the common phase error of a plurality of pilot carriers in a second symbol and the common phase error of corresponding pilot carriers in the first symbol. The output of the accumulator is filtered, and coupled to the I/Q demodulator.

According to another aspect of the invention the coupled output of the accumulator of the automatic frequency control circuit is enabled in the I/Q demodulator only during reception of a guard interval therein.

According to yet another aspect of the invention the channel estimation and correction circuitry also has an automatic sampling rate control circuit coupled to the phase extraction circuit, in which a memory stores the individual accumulated phase errors of pilot carriers in a first symbol carried in the phase-uncorrected I and Q data. An accumulator is coupled to the memory and accumulates differences between the

phase errors of individual pilot carriers in a second symbol and phase errors of corresponding pilot carriers in the first symbol to define a plurality of accumulated intersymbol carrier phase error differentials. A phase slope is defined by a difference between a first accumulated intersymbol carrier phase differential and a second accumulated intersymbol carrier phase differential. The output of the accumulator is filtered and coupled to the I/Q demodulator.

According to one aspect of the invention the sampling rate control circuit stores a plurality of accumulated intersymbol carrier phase error differentials and computes a line of best fit therebetween.

According to another aspect of the invention the coupled output signal of the accumulator of the automatic sampling rate control circuit is enabled in the resampling circuit only during reception of a guard interval therein.

According to an aspect of the invention a common memory for storing output of the phase extraction circuit is coupled to the automatic frequency control circuit and to the automatic sampling rate control circuit.

According to another aspect of the invention the phase extraction circuit also has a pipelined circuit for iteratively computing the arctangent of an angle of rotation according to the series

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots, \qquad |x| < 1$$

wherein x is a ratio of the phase-uncorrected I and Q data.

The pipelined circuit includes a constant coefficient multiplier, and a multiplexer for selecting one of a plurality of constant coefficients of the series. An output of the multiplexer is connected to an input of the constant coefficient multiplier.

According to still another aspect of the invention the pipelined circuit has a multiplier, a first memory for storing the quantity x^2, wherein the first memory is coupled to a first input of the multiplier, and has a second memory for holding an output of the multiplier. A feedback connection is provided between the second memory and a second input of the multiplier. The pipelined circuit also has a third memory for storing the value of the series. Under direction of a control circuit coupled to the third memory, the pipeline circuit computes N terms of the series, and also computes N+1 terms of the series. An averaging circuit is also coupled to the third memory and computes the average of N terms and N+1 terms of the series.

Data transmitted in a pilot carrier of the multicarrier signal is BCH encoded according to a code generator polynomial h(x). A demodulator operative on the BCH encoded data is provided, which includes an iterative pipelined BCH decoding circuit.

The BCH decoding circuit is circuit coupled to the demodulator. It forms a Galois Field of the polynomial, and calculates a plurality of syndromes therewith. The BCH decoding circuit includes a plurality of storage registers, each storing a respective one of the syndromes, and a plurality of feedback shift registers, each accepting data from a respective one of the storage registers. The BCH decoding circuit has a plurality of Galois field multipliers. Each of the multipliers is connected in a feedback loop across a respective one of the feedback shift registers and multiplies the output of its associated feedback shift register by an alpha value of the Galois Field. An output Galois field multiplier multiplies the outputs of two of the feedback shift registers.

A logical network forms an error detection circuit connected to the feedback shift registers and to the output Galois field multiplier. The output of the error detection circuit indicates an error in a current bit of data, and a feedback line is enabled by the error detection logic and connected to the storage registers. Using the feedback line, the data output by the feedback shift registers are written back into the storage registers for use in a second iteration.

According to an aspect of the invention the output Galois field multiplier has a first register initially storing a first multiplicand A, a constant coefficient multiplier connected to the first register for multiplication by a value α. An output of the constant coefficient multiplier is connected to the first register to define a first feedback loop, whereby in a kth cycle of clocked operation the first register contains a Galois field product $A\alpha^k$. A second register is provided for storing a second multiplicand B. An AND gate is connected to the second register and to the output of the constant coefficient multiplier. An adder has a first input connected to an output of the AND gate. An accumulator is connected to a second input of the adder, and the Galois field product AB is output by the adder.

The invention provides a method for the estimation of a frequency response of a channel. It is performed by receiving from a channel an analog multicarrier signal that has a plurality of data carriers and scattered pilot carriers. The scattered pilot carriers are spaced apart at an interval N and are transmitted at a power that differs from the transmitted power of the data carriers. The analog multicarrier signal is converted to a digital representation thereof. A Fourier transform is performed on the digital representation of the multicarrier signal to generate a transformed digital signal. The bit order of the transformed digital signal is reversed to generate a bit-order reversed signal. Magnitudes of the carriers in the bit-order reversed signal are cyclically accumulated in N accumulators, amd the accumulated magnitudes are correlated with the power of the scattered pilot carriers. Responsive to the correlation, a synchronizing signal is

generated that identifies a carrier position of the multicarrier signal, preferably an active carrier.

According to another aspect of the invention the step of accumulating magnitudes is performed by adding absolute values of a real component of the bit-order reversed signal to respective absolute values of imaginary components thereof to generate sums, and respectively storing the sums in the N accumulators.

According to yet another aspect of the invention the step of correlating the accumulated magnitudes also is performed by identifying a first accumulator having the highest of the N values stored therein, which represents a first carrier position, and by identifying a second accumulator which has the second highest of the N values stored therein, which represents a second carrier position. The interval between the first carrier position and the second carrier position is then determined.

To validate the consistency of the carrier position identification, the position of a carrier of a first symbol in the bit-order reversed signal is compared with a position of a corresponding carrier of a second symbol therein.

Preferably interpolation is performed between pilot carriers to determine correction factors for respective intermediate data carriers disposed therebetween, and respectively adjusting magnitudes of the intermediate data carriers according to the correction factors.

According to an aspect of the invention a mean phase difference is determined between corresponding pilot carriers of successive symbols of the transformed digital signal. A first control signal representing the mean phase difference, is provided to control the frequency of reception of the multicarrier signal. The first control signal is enabled only during reception of a guard interval.

Preferably a line of best fit is determined for the inter-symbol phase differences of multiple carriers to define a phase slope.

For a better understanding of these and other objects of the present invention, reference is made to the detailed description of the invention, by way of example, which is to be read in conjunction with the following drawings, wherein:

Fig. 1 illustrates the spectrum of a COFDM subchannel;

Fig. 2 shows a frequency spectrum for multiple carriers in a COFDM signal;

Fig. 3 is a diagram of a signal according to COFDM and shows a data symbol format;

Fig. 4 is a block diagram illustrating an FFT based COFDM system;

Fig. 5 illustrates certain perturbations in a COFDM signal constellation;

Fig. 6 is a flow diagram of a method of timing synchronization according to a preferred embodiment of the invention;

Fig. 7 is a plot of an F ratio test performed on several data symbols for coarse timing synchronization;

Fig. 8 is a plot of an incomplete beta function for different degrees of freedom;

Fig. 9 is a plot helpful in understanding a test of statistical significance according to the invention;

Fig. 10 is an electrical schematic of a synchronization circuit according to an alternate embodiment of the invention;

Fig. 11 is an electrical schematic of a synchronization circuit according to another alternate embodiment of the invention;

Fig. 12 is a block diagram of a single-chip embodiment of a digital receiver in accordance with the invention;

Fig. 13 is a block diagram illustrating the front end of the digital receiver shown in Fig. 12 in further detail;

Fig. 14 is a block diagram illustrating the FFT circuitry, channel estimation and correction circuitry of the digital receiver shown in Fig. 12;

Fig. 15 is a block diagram illustrating another portion of the digital receiver shown in Fig. 12;

Fig. 16 is a more detailed block diagram of the channel estimation and correction circuitry shown in Fig. 14;

Fig. 17 is a schematic of the automatic gain control circuitry of the digital receiver shown in Fig. 12;

Fig. 18 is a schematic of the I/Q demodulator of the digital receiver shown in Fig. 12;

Fig. 19 illustrates in greater detail a low pass filter shown in Fig. 13;

Fig. 20 shows the response of the low pass filter shown in Fig. 19;

Fig. 21 shows the resampling circuitry of the digital receiver shown in Fig. 12;

Fig. 22 illustrates a portion of an interpolator in the resampling circuitry of Fig. 21;

Fig. 23 is a more detailed block diagram of the FFT window circuitry shown in Fig. 14;

Fig. 24 is a schematic of a butterfly unit in the FFT calculation circuitry shown in Fig.14;

Figs. 25 and 26 are schematics of butterfly units in accordance with the prior art;

Fig. 27 is a schematic of a radix $2^2 + 2$ FFT processor in accordance with the invention;

Fig. 28 is 32 point flow graph of the FFT processor shown in Fig. 27;

Fig. 29 is a schematic of a configurable 2K/8K radix 2^2+2 single path, delay feedback pipelined FFT processor in accordance with the invention;

Fig. 30 is a detailed schematic of a complex multiplier used in the circuitry shown in Fig. 29;

Fig. 31 is a detailed schematic of an alternate embodiment of a complex multipliers used in the circuitry shown in Fig. 29;

Fig. 32 is another diagram illustrating the organization of the twiddle factors for each of the multipliers in the circuitry shown in Fig. 29;

Fig. 33 illustrates the organization of the twiddle factors for each of the multipliers in the circuitry shown in Fig. 29;

Fig. 34 is a schematic of address generator used in the circuitry shown in Fig. 29;

Fig. 35 is a schematic of a generalization of the address generator shown in Fig. 34;

Fig. 36 is a flow chart illustrating the process of pilot location conducted by the channel estimation and correction circuitry shown in Fig. 16;

Fig. 37 is a flow chart of an embodiment of the pilot localization procedure according to the invention.

Fig. 38 is a more detailed block diagram of the tps sequence block of the circuitry shown in Fig. 14;

Fig. 39 is a schematic of a BCH decoder used in the tps processing circuitry shown in Fig. 38;

Fig. 40 is a more detailed schematic of a Galois field multiplier shown in Fig. 39;

Fig. 41 is a block diagram generically illustrating the automatic sampling control and automatic frequency control loops of the digital receiver shown in Fig. 12;

Fig. 42 is a more detailed block diagram of the automatic sampling control and automatic frequency control loops shown in Fig. 41;

Fig. 43 is a more detailed block diagram of the phase extract block of the circuitry shown in Fig. 42;

Fig. 44 is a schematic of the circuitry employed to calculate an arctangent in the block diagram shown in Fig. 43;

Fig. 45 is a plot of the square error at different values of α of the Taylor expansion to 32 terms;

Fig. 46 is a plot of the square error at different values of α of the Taylor expansion to 31 terms;

Fig. 47 is a plot of the square error at different values of α of the average of the Taylor expansion to 31 and 32 terms;

Fig. 48 is a plot of the phase differences of pilot carriers with a line of best fit shown;

Fig. 49 is a more detailed block diagram an alternate embodiment of the automatic sampling control and automatic frequency control loops shown in Fig. 41;

Fig. 50 illustrates a coded constellation format used in the demapping circuitry of Fig. 15;

Fig. 51 illustrates the conversion of I,Q data to binary data value using the format shown in Fig. 50;

Fig. 52 is a more detailed block diagram of the symbol deinterleaving circuitry shown in Fig. 15;

Fig. 53 is a more detailed block diagram of the bit deinterleaving circuitry shown in Fig. 15;

Fig. 54 illustrates the conversion from a coded constellation format to a 24 bit soft I/Q format by the bit deinterleaving circuitry shown in Fig. 53;

Fig. 55 is a more detailed block diagram of the microprocessor interface of the receiver shown in Fig. 12;

Fig. 56 is a more detailed block diagram of the system controller of the receiver shown in Fig. 12; and

Fig. 57 is a state diagram relating to channel acquisition in the system controller of the receiver shown in Fig. 56.

Alignment of The FFT Window

Referring again to Figs. 3 and 4, according to the invention a statistical method is applied to COFDM signals to find the end of the guard interval 10. This method is explained with reference to the above noted European Telecommunications Standard, but is applicable to many forms of frequency division multiplexing having prefixed or postfixed guard intervals. It allows the receiver 32 to find the end of the guard interval given only the received sampled complex signal (solid line 12) and the size of the active interval 6 . The method relies on the fact that the guard interval 10 is a copy of the last part of the data symbol 4. In the receiver 32, due to echoes and noise from the channel and errors in the local oscillator, the guard interval 10 and the last part of the data symbol 4 will differ. If the errors introduced are random then a statistical method can be applied. According to the invention, the received complex signal is sampled at a rate which is nearly identical to that used in the transmitter. A difference signal is found for a pair of received samples which are separated by a period of time which is as close as possible to the active interval 6. This period should be equal to the size of the fast fourier transform ("FFT") being applied (i.e. 2048 or 8192 samples). Let

$$S_i = |s_i| - |s_{i-fftsize}| \qquad (14)$$

where S_i is the difference signal; s_i and $s_{i\text{-fftsize}}$ are the current and previous complex input samples of which the modulus is taken. That is, the subscript "i" indexes a linear time sequence of input values. Assuming that the input signal is random, then S_i is also random. Within the guard interval s_i and $s_{i\text{-fftsize}}$ will be similar, although not identical, due to the effects of the channel. S_i will be therefore a random signal with a small dispersion. As used herein the term "dispersion" means generally the spread of values, and is not restricted to a particular mathematical definition. In general the active part of one symbol is not related to the active part of the next symbol. Outside of the guard interval S_i will be random with a much larger dispersion. In order to find the end of the guard interval, the dispersion of the difference signal S_i is monitored to look for a significant increase which will occur at the boundary of the guard interval 10 and the active interval 6. The inventors have also observed that a large decrease in dispersion is seen at the start of the guard interval 10.

According to a preferred embodiment of the invention samples of the input signal are stored over an interval which includes at least one symbol period T_s. The dispersion of the difference signal S_i is calculated over a block of samples. The block is moved back in time over a number of samples, n, and the dispersion is recalculated. These two blocks are referred to herein as "comparison blocks". The ratio of a current dispersion in a first comparison block to the dispersion in a previous comparison block is found. Then, the F ratio significance test is used to find significant differences in the dispersions of the two comparison blocks. The F ratio is defined as

$$F = \frac{\mathrm{VAR}(i)}{\mathrm{VAR}(i-n)} \qquad (15)$$

where n is a positive integer, i indexes the input samples, and VAR(i) is the variance of a block of values of length N samples. Variance can be defined as

$$\mathrm{VAR}(i) = \frac{1}{N}\sum_{j=0}^{N}(S_{i-j})^2 - \left(\frac{1}{N}\sum_{j=0}^{N} S_{i-j}\right)^2 \qquad (16)$$

While the F ratio significance test is used in the preferred embodiment, other functions of the two dispersion values which give a signal relating to the change in dispersion could be used. There are many such functions. An advantage of the F ratio is that for a random input signal it has a known probability distribution, allowing convenient statistical analysis for purposes of performance analysis and system design. Also the F ratio intrinsically normalizes the signal, making the result independent of the signal level.

The method is disclosed with reference to Fig. 6, in which a first member of a sample pair in a current evaluation block is measured at step 38. A delay of one active interval 6 (Fig. 3) is experienced in step 40. This may be accomplished with a digital delay such as a FIFO, or equivalently by buffering samples for an active interval in a memory and accessing appropriate cells of the memory. A second member of the sample pair is measured in step 42, and the difference between the first and second member is determined and stored in step 44. The end of the current block is tested at decision step 46. The size of the evaluation block should not exceed the length of a guard interval, and may be considerably smaller. In the event the end of the current block has not yet been reached, another sample is acquired at step 48, and control returns to step 38.

If the end of the current block has been reached, the dispersion of the current block is measured in step 50, and is treated as one of two comparison blocks of data. A test is made at decision step 52 to determine if a group of two comparison blocks have been evaluated. If this test is negative, then another block of data is acquired in step 54, after which control returns to step 38. The other block of data need not be contiguous with the block just completed.

In the event the test at decision step 52 is positive, the F ratio is computed for the group of two comparison blocks at step 56. The results obtained in step 56 are submitted to peak detection in step 60. Peak detection optionally includes statistical tests of significance, as is explained hereinbelow.

If peaks are detected, then the boundary of a guard interval is established in step 62 for purposes of synchronization of the FFT window which is necessary for further signal reconstruction. If peaks are not detected, the above process is repeated with a block of samples taken from another portion of the data stream.

Example 1:

Referring now to Fig. 7 a complex signal was generated according to the above noted European Telecommunications standard using a random number generator, and transmitted across a Ricean channel model together with added white Gaussian noise (SNR = 3.7). Data symbols were then analyzed according to the above described method. The results 6 data symbols are shown in Fig. 7, wherein the F ratio is plotted for convenience of presentation on a logarithmic axis as line 64, because the spikes 66, 68, at the beginning and end of the guard intervals respectively, are very large.

Although it is quite evident from Figure 7 that the ends of the guard intervals are easy to find using any of several well known peak detectors, it is possible to apply a statistical test to more accurately answer the question: do the two blocks of samples have the same dispersion? This is the null hypothesis, H_0, i.e. the dispersion is the

same and the observed spike in F is due to random fluctuations only. If H_0 has very low probability it can be rejected, which would correspond to detection of the start or end of the guard interval. From the way the COFDM symbol is constructed H_0 is expected to be true for comparison blocks lying entirely within the guard interval or within the active interval, but false when the comparison blocks straddle a boundary at the start or end of the guard interval. If comparison blocks of random samples are drawn from the same population then the probability of F is given by

$$Q(F|v_1,v_2)=I_x(\frac{v_1}{2},\frac{v_2}{2}) \quad \textbf{(17)}$$

where I() is the incomplete Beta function,

$$x = \frac{v_2}{v_2 + v_1 F} \quad \textbf{(18)}$$

and v_1 and v_2 are the number of degrees of freedom with which the first and second dispersions are estimated. In this example v1 = v2 = (N-1) if n >= N. The shape of the function is shown in Fig. 8. From a statistical point of view n should be sufficiently large so that the two blocks do not overlap, i.e. n >= N. If the blocks do overlap, then the calculation of the second dispersion will use samples used for the calculation of the first dispersion. This effectively reduces the number of degrees of freedom and hence the significance of the result. It has been determined that setting n=N works well.

The function Q() in equation (13) actually gives the one-tailed probability. H_0 could be rejected if F is either very large or very small, and so the two-tailed test is required. Actually the two tails are identical, so for a two-tailed test the probability is double that given in equation (13). However, this results in values of probability greater than one for F<1. The probability, p, is therefore calculated as follows:

$$p=2I_x(\frac{v_1}{2},\frac{v_2}{2}) \quad \textbf{(19)}$$

and then, if (p > 1), p = 2 - p. This probability reflects the viability of H_0. Thus if p is small, H_0 can be rejected and it can be stated, with a specified degree of certainty, that the comparison blocks come from sample populations with different dispersion. The noted European Telecommunications Standard specification states that the block size, N, should be 32 for a correlation algorithm. N={32,64} have been successfully tried. The probability functions obtained are shown in Fig. 9 using these values for N. In the preferred embodiment p <= 0.05 has been set for the rejection of H_0.

A precise implementation would be to calculate F, then x, then the incomplete Beta function, then p and then apply the threshold test. This algorithm would be very difficult to realize in hardware since the Beta function is very complicated. In the preferred embodiment it is much simpler, and gives the same results, to set the acceptance threshold and N parameter, and thus define an upper and lower limit for F. It is then only necessary to calculate F and compare it with the limits. In order to simply find the end of the guard interval it may be safely assumed that F>1. Only the upper limit on F is needed. To calculate the limits on F accurately, a suitable root-finding method, such as Newton-Raphson may be utilized. Typical values are given in Table 1.

Table 1

p threshold	v1 = v2 = 31		v1 = v2 = 63	
	F_lower	F_upper	F_lower	F_upper
0.2	0.627419	1.593832	0.722591	1.383909
0.1	0.548808	1.822132	0.658620	1.518326
0.05	0.488143	2.048582	0.607525	1.646022
0.01	0.386894	2.584689	0.518205	1.929738
0.005	0.354055	2.824422	0.487936	2.049448
0.001	0.293234	3.410251	0.429794	2.326695
10^{-4}		4.337235		
10^{-5}		5.393528		
10^{-6}		6.605896		
10^{-7}		8.002969		
10^{-8}		9.616664		

This method has been successfully tested using the specified channel model with additive white Gaussian noise (SNR=3.7).

The formula for dispersion given in Equation (12) would require a multiplier for implementation in silicon. The calculation of F is a division in which the (N-1) normalisation constants cancel out as long as the two blocks have the same size. Accurate multiplication and division can be expensive in silicon. In the preferred embodiment simplifications have been implemented which give less accurate, but still viable, values for F. S_i can be assumed to have zero mean so it is not necessary to calculate the mean from the block of samples. This also increases the number of degrees of freedom from (N-1) to N. Instead of calculating variance using the standard

sum of squares formula, the dispersion can be estimated by the mean absolute deviation. The formula for VAR(i) becomes

$$VAR(i) = \frac{1}{N}\left(\sum_{j=0}^{N-1} |s_{i-j}|\right)^2 \qquad (20)$$

The (1/N) factor divides out in the calculation of F if the two blocks have the same size. But there still remains the division of the two dispersions and the squaring required. These can be tackled using logarithms to the base 2. Substituting from Equation (16) into Equation (11) gives

$$F = \left(\frac{\sum_{j=0}^{N-1} |s_{i-j}|}{\sum_{j=0}^{N-1} |s_{i-n-j}|}\right)^2 = \left(\frac{s_a}{s_b}\right)^2 \qquad (21)$$

Taking logs to the base 2 gives

$$\log F = 2(\log s_a - \log s_b) = y \qquad (22)$$

It is then only necessary to calculate y and compare it with the logarithm to the base 2 of the F upper limit. The comparison can be made by subtracting the log of the limit from 2(log2sa-log2sb) and comparing with zero. The factor of 2 can be absorbed into the limit.

Calculation of the logs to base two is relatively straightforward in hardware if the numbers are stored as fixed point fractions. The fractions can be split into an exponent and a fractional mantissa: $x = A2^B$. Taking log base 2 gives logx = logA + B. Since A is fractional it is practical to find its logarithm using a lookup table. The exponent B can be found from the position of the MSB (since s_a and s_b will both be positive numbers).

The calculation can thus be reduced to require only addition and subtraction arithmetic operations. The limit should also be recalculated using v1=v2=N if using this method. In practice, the significance level may be set empirically for a particular application, preferably p = 0.05.

It will be appreciated by those skilled in the art that various measures of dispersion may be utilized without departing from the spirit of the invention, for example the

standard deviation, skew, various moments, histograms, and other calculations known in the art.

In a first alternate embodiment of the invention, the above described method is employed using either the real or the imaginary parts of the signal instead of the modulus. This embodiment achieves economy in hardware.

In a second alternate embodiment of the invention, the n parameter of equation (11) has been optimized. At the end of the guard interval, the two blocks straddle more of the transition to the active interval, giving a well-defined increase in the dispersion. Using any value n>2 has the drawback that several successive points will give significant increases as the later block travels up to the boundary. This small problem is easily overcome by introducing a dead period after detection of the boundary. That is, once a spike has been detected a set of samples equal to the size of the FFT window is accepted before further attempts are made to locate another spike. The dead period has the added benefit of not introducing false spikes. When using larger values of n the spikes 66, 68 (Fig. 7) increase, whilst the H_0 noisy F signal remain much the same.

Example 2:

The maximum F-spike height as a function of n has been measured systematically together with the background variation in F. The results are shown in Table 2.

Table 2

(1)	(2)	(3)	(4)	(5)
n	<F>	$F_{s.d.}$	F_{max}	(4) / (3)
3	1.0009	0.07	7.5	107
5	1.0012	0.10	10.7	107
10	1.0011	0.14	12.9	92
15	1.0014	0.17	16.7	98
20	1.0014	0.19	19.3	102
30	1.0012	0.23	20.9	91
40	0.9975	0.24	22.0	92
50	0.9926	0.25	20.4	81.6

Table 2 was developed using the first 5 frames of the signal analyzed in Fig. 7. The statistics in columns (2) and (3) of Table 2 were made by excluding any points where F>=3.0 to exclude spikes from the calculations. The spikes would otherwise affect the values of mean and standard deviation even though they are from a different statistical population.

The results indicate that the background variation in F, $F_{s.d.}$, was affected by n, increasing asymptotically to a value of approximately 0.28. It is likely that this is the effect of overlapping blocks. For example, for N=64 and n<64, the blocks over which the dispersions are calculated will contain some of the same values and therefore be correlated. To test this theory Fs.d. was evaluated for n>N, and the results are shown in Table 3.

Table 3

n	$F_{s.d.}$
60	0.258
70	0.266
80	0.270
90	0.278
100	0.285
128	0.297
256	0.366

The dependence becomes linear at n >= N/2. If F is calculated every n samples, rather than every sample, then this dependence may be reduced. However, this creates a risk for small guard intervals of not having the first block wholly within the guard interval and the second wholly within the active interval.

A third alternate embodiment of the invention is disclosed with reference to Fig. 10, which schematically illustrates a timing synchronization circuit 70. The circuit accepts a complex input signal 72, and includes a circuit module 74 which develops the modulus of its input, which is taken from node 83. The circuit module 74 insures that the value being subsequently processed is an unsigned number. The input to the circuit module 74 is a difference signal which is developed by a subtracter 75 which takes as inputs the input signal 72 and a delayed version of the input signal 72 which has been processed through a delay circuit 79, preferably realized as a FIFO 77 of length L, where L is the size of the FFT window. As explained above, it is also possible to operate this circuit where the input signal 72 is real, imaginary, or complex, or even the modulus of a complex number. In the case where the input signal 72 is real, or imaginary, the circuit module 74 can be modified, and can be any known circuit that removes the sign of the output of the subtracter 75, or equivalently sets the sign so that the outputs accumulate monotonically; i.e. the circuit has a unipolar output. The output of the circuit module 74 is ultimately clocked into a digital delay, which is preferably implemented as a FIFO 78. When the FIFO 78 is full, a signal SIG1 80 is asserted, and the output of the FIFO 78

becomes available, as indicated by the AND gate 82. An adder/subtracter circuit 84 is also connected to the node 76, and its output is stored in a register 86. A delayed version of the output of the adder/subtracter circuit 84 is taken from the register 86 and fed back as a second input to the adder/subtracter circuit 84 on line 88. In the event the signal SIG1 80 has been asserted, a version of the output of the circuit module 74, delayed by a first predetermined interval N, where N is the number of samples in the comparison blocks, is subtracted from the signal on node 76.

The signal on line 88 is an index into a lookup table, preferably implemented as a read-only-memory ("ROM"), and shown as ROM 90. The address of the ROM 90 contains the logarithm to the base 2 of the magnitude of the signal on line 88, which then appears at node 92. The node 92 is connected to a subtracter 94, and to a delay circuit, shown as FIFO 98, which is used to develop the denominator of the middle term of equation (17).

The subtracter 94 produces a signal which is compared against the $\log_2$ of a predetermined threshold value F_{LIMIT} in a comparison circuit 106, shown for simplicity as an adder 108 connected to a comparator 110. The output signal SYNC 112 is asserted when the boundary of a guard interval has been located.

Although not implemented in the presently preferred embodiment, It is also possible to configure the size of the FIFO 77 dynamically, so that the size of the interval being evaluated can be adjusted according to operating conditions. This may conveniently be done by storing the values on the node 92 in a RAM 114 for computation of their dispersion.

In a fourth alternate embodiment of the invention, explained with reference to Fig. 11, components similar to those of the embodiment shown in Fig. 10 have the same reference numerals. A timing synchronization circuit 116 is similar to the timing synchronization circuit 70, except now the delay circuit 79 is realized as the FIFO 77, and another FIFO 100, one of which is selected by a multiplexer 102. Both of the FIFOs 77, 100 provide the same delay; however the capacities of the two are different. The FIFO 100 provides for storage of samples taken in an interval equal to the size of the FFT window, and is normally selected in a first mode of operation, for example during channel acquisition, when it is necessary to evaluate an entire symbol in order to locate a boundary of a guard interval. In the noted European Telecommunications standard, up to 8K of data storage is required, with commensurate resource requirements. During subsequent operation, the approximate location of the guard interval boundaries will be known from the history of the previous symbols. In a second mode of operation, It is therefore only necessary to evaluate a much smaller interval in order to verify the exact location of the guard interval boundary. The number of samples used in the computation

of the dispersion can be kept to a small number, preferably 32 or 64, and the much smaller FIFO 77 accordingly selected to hold the computed values. The resources saved thereby can be utilized for other functions in the demodulator, and memory utilized by the larger FIFO 100 may also be reallocated for other purposes.

A control block 81 optionally advances the evaluation interval relative to symbol boundaries in the data stream in successive symbols, and can also be used to delay for the dead period. Eventually the moving evaluation interval straddles the boundary of the current symbol's guard interval, and synchronization is then determined. The size of the evaluation interval is chosen to minimize the use of memory, yet to be large enough to achieve statistical significance in the evaluation interval. The size of the evaluation interval, and the FIFO 77 may be statically or dynamically configured.

Single Chip Implementation of a COFDM Demodulator

Overview

Referring initially to Fig. 12, there is shown a high level block diagram of a multicarrier digital receiver 126 in accordance with the invention. The embodiment described hereinbelow conforms to the ETS 300 744 telecommunications standard (2K mode), but can be adapted by those skilled in the art to operate with other standards without departing from the spirit of the invention. A radio frequency signal is received from a channel such as an antenna 128, into a tuner 130, which is conventional, and preferably has first and second intermediate frequency amplifiers. The output of the second intermediate frequency amplifier (not shown), is conducted on line 132 to an analog to digital converter 134. The digitized output of the analog to digital converter 134 is provided to block 136 in which I/Q demodulation, FFT, channel estimation and correction, inner and outer deinterleaving, and forward error correction are conducted. Carrier and timing recovery are performed in block 136 entirely in the digital domain, and the only feedback to the tuner 130 is the automatic gain control ("AGC") signal which is provided on line 138. A steady 20 MHz clock on line 140 is provided for use as a sampling clock for the external analog to digital converter 134. A host microprocessor interface 142 can be either parallel or serial. The system has been arranged to operate with a minimum of host processor support. In particular channel acquisition can be achieved without any host processor intervention.

The functions performed within the block 136 are grouped for convenience of presentation into a front end (Fig. 13), FFT and channel correction group (Fig. 14), and a back end (Fig. 15).

As shown in Fig. 13, I/Q samples at are received by an IQ demodulator 144 from the analog to digital converter 134 (Fig. 12) on a bus 146 at a rate of 20 megasamples per second. An AGC circuit 148 also takes its input from the bus 146. A frequency rate

control loop is implemented using a numerically controlled oscillator 150, which receives frequency error signals on line 152, and frequency error update information on line 154. Frequency and sampling rate control are achieved in the frequency domain, based on the pilot carrier information. The frequency error signals, which are derived from the pilot carriers, and the frequency error update information will both be disclosed in further detail shortly. The I and Q data output from the IQ demodulator 144 are both passed through identical low pass filters 156, decimated to 10 megasamples per second, and provided to a sinc interpolator 158. Sample rate control is achieved using a numerically controlled oscillator 160 which receives sample rate control information derived from the pilot signals on line 162, and receives sample error update timing information on line 164.

As shown in Fig. 14, acquisition and control of the FFT window are performed in block 166, which receives signals from the sinc interpolator 158 (Fig. 13). The FFT computations are performed in FFT calculation circuitry 168. Channel estimation and correction are performed in channel estimation and correction block 170, and involves localization of the pilot carriers, as will be described below in greater detail. The tps information obtained during pilot localization is processed in tps sequence extract block 172. Uncorrected pilot carriers are provided by the circuitry of channel estimation and correction block 170 to correction circuitry 174, which develops sampling rate error and frequency error signals that are fed back to the numerically controlled oscillators 150, 160 (Fig. 13).

Referring to Fig. 15, corrected I and Q data output from channel estimation and correction block 170 are provided to demapping circuitry 176. The current constellation and hierarchical constellation parameters, derived from the tps data, are also input on lines 178, 180. The resulting symbols are deinterleaved in symbol deinterleaver 182, utilizing a 1512 x 13 memory store. One bit of each cell in the memory store is used to flag carriers having insufficient signal strength for reliable channel correction. Bit deinterleaver 184 then provides deinterleaved I and Q data to a Viterbi Decoder 186, which discards the flagged carriers, so that unreliable carriers do not influence traceback metrics. A Forney deinterleaver 188 accepts the output of the Viterbi Decoder 186 and is coupled to a Reed-Solomon decoder 190. The forward error correction provided by the Viterbi and Reed-Solomon decoders is relied upon to recover lost data in the case of flagged carriers.

Referring to Fig. 16, in the presently preferred embodiment a mean value is calculated in block 192 for uncorrected carriers with reference to the previous symbol. Data carriers whose interpolated channel response falls below some fraction, preferably 0.2, of this mean will be marked with a bad_carrier flag 194. The bad_carrier flag 194

is carried through the demapping circuitry 176, symbol deinterleaver 182, and bit deinterleaver 184, to the Viterbi Decoder 186 where it is used to discard data relating to the unreliable carriers. The parameters used to set the bad_carrier flag 194 can be varied by the microprocessor interface 142.

An output interface 196 produces an output which can be an MPEG-2 transport stream. The symbol deinterleaver 182, and the bit deinterleaver 184 are conventional. The Viterbi decoder 186, Forney deinterleaver 188, Reed-Solomon decoder 190, and the output interface 196 are conventional. They can be the components disclosed in copending Application No. 638,273, entitled "An Error Detection and Correction System for a Stream of Encoded Data", filed April 26, 1996, Application No. 480,976, entitled "Signal Processing System", filed June 7, 1995, and Application No. 481,107, entitled "Signal Processing Apparatus and Method", filed June 7, 1995, all of which are commonly assigned herewith, and are incorporated herein by reference. The operation of the multicarrier digital receiver 126 (Fig. 12) is controlled by a system controller 198.

Optionally the hierarchical constellation parameters can be programmed to speed up channel acquisition, rather than derived from the tps data.

The input and output signals and the register map of the multicarrier digital receiver 126 are described in tables 4, and 5 respectively.

Automatic Gain Control

The purpose of the AGC circuit 148 (Fig. 13)is to generate a control signal to vary the gain of the COFDM input signal to the device before it is analog-to-digital converted. As shown in greater detail in Fig. 17, a Sigma-Delta modulator 200 is used to provide a signal which can be used as a gain control to a tuner, once it has been low-pass filtered by an external R-C network.

The magnitude of the control voltage signal 202 is given by:

$$\text{control\_voltage} = \text{control\_voltage} + \text{error} \qquad \textbf{(23)}$$

where

$$\text{error} = K(|\text{data}| - \text{mean}) \qquad \textbf{(24)}$$

where K is a constant (normally K<<1) which determines the gain in the AGC control loop. The mean value can be determined from the statistics of Gaussian noise, which is a close approximation to the properties of the COFDM input signal, where the input data is scaled to +/-1. The control voltage signal 202 is set back to its initial value when the signal resync 204 is set low, indicating a channel change or some other event requiring resynchronization.

The input and output signals and the registers for the microprocessor interface 142 of the AGC circuit 148 are described in tables 6, 7, and 8 respectively.

IQ Demodulator

The function of the IQ demodulator 144 (Fig. 13) is to recover in-phase and quadrature components of the received sampled data. It is shown in further detail in Fig. 18.

The numerically controlled oscillator 150 generates in-phase and quadrature sinusoids at a rate of (32/7) MHz, which are multiplied with data samples in multipliers 206. The address generator 208 advances the phase linearly. The frequency error input 210 increments or decrements the phase advance value. The samples are multiplied with the sinusoids in the multipliers 206using 10 bit x 10 bit multiply operations. In one embodiment the IQ demodulator 144 is operated at 20 MHZ and then retimed to 40MHz in retiming block 212. In a preferred embodiment the IQ demodulator 144 is operated at 40MHz, in which case the retiming block 212 is omitted.

Sinusoids are generated by the address generator 208 on lines 214, 216. The phase value is employed as an address into a lookup table ROM 218. Only quarter cycles are stored in the lookup table ROM 218 to save area. Full cycles can be generated from the stored quarter cycles by manipulating the data from the ROM 218 and inverting the data in the case of negative cycles. Two values are read from the lookup table ROM 218 for every input sample -- a cosine and a sine, which differ in phase by 90 degrees.

The input and output signals of the IQ demodulator 144 are described in tables 9 and 10 respectively.

Low Pass Filter

The purpose of the low pass filters 156 (Fig. 13) is to remove aliased frequencies after IQ demodulation - frequencies above the 32/7 MHz second IF are suppressed by 40dB. I and Q data are filtered separately. The output data is decimated to 10 megasamples per second ("Msps") because the filter removes any frequencies above 1/4 of the original 20 Msps sampling rate. The filter is constructed with approximately 60 taps which are symmetrical about the center, allowing the filter structure to be optimized to reduce the number of multipliers 220. Fig. 19 is a block diagram of one of the low pass filters 156, the other being identical. Fig. 19 shows a representative symmetrical tap 222, and a center tap 224. The required filter response of the low pass filters 156 is shown in Fig. 20.

The input and output signals of the low pass filters 156 are described in tables 11 and 12 respectively.

Resampling

Referring to Fig. 13, the purpose of resampling is to reduce the 10 Msps data stream output from the low pass filters 156 down to a rate of (64/7) Msps, which is the

nominal sample rate of the terrestrial digital video broadcasting ("DVB-T") modulator at the transmitter.

Resampling is accomplished in the sinc interpolator 158, and the numerically controlled oscillator 160. The latter generates a nominal 64/7 MHZ signal. The resampling circuitry is shown in further detail in Fig. 21. The numerically controlled oscillator 160 generates a valid pulse on line 226 and a signal 228 representing the interpolation distance for each 40MHz clock cycle in which a 64/7MHz sample should be produced. The interpolation distance is used to select the appropriate set of interpolating filter coefficients which are stored in coefficient ROMs 230. It should be noted that only the sinc interpolator for I data is illustrated in Fig. 21. The structures for Q data are identical.

Fig. 22 illustrates the generation of the interpolation distance and the valid pulse. Nominally T_s = 1/10 Msps, and T = 1/ (64/7) Msps. The sinc interpolation circuit disclosed in our noted Application No. 08/638,273 is suitable, with appropriate adjustment of the operating frequencies.

The input and output signals of the sinc interpolator 158 and the numerically controlled oscillator 160 are described in tables 13 and 14 respectively.

FFT Window

As has been explained in detail above, the function of the FFT Window function is to locate the "active interval" of the COFDM symbol, as distinct from the "guard interval". This function is referred to herein for convenience as "FFT Window". In this embodiment the active interval contains the time domain representation of the 2048 carriers which will be recovered by the FFT itself.

The FFT window operates in two modes; Acquisition and Tracking. In Acquisition mode the entire incoming sample stream is searched for the guard interval/active interval boundary. This is indicated when the F-ratio reaches a peak, as discussed above. Once this boundary has been located, window timing is triggered and the incoming sample stream is searched again for the next guard interval/active interval boundary. When this has been located the length of the guard interval is known and the expected position of the next guard/active boundary can be predicted. The FFT window function then switches to tracking mode.

This embodiment is similar to the fourth alternate embodiment discussed above in respect of the tracking mode. In tracking mode only a small section of the incoming sample stream around the point where the guard/active boundary is expected to be is searched. The position of the active interval drifts slightly in response to IF frequency and sampling rate offsets in the front-end before the FFT is calculated. This drift is

tracked and FFT window timing corrected, the corrections being inserted only during the guard interval.

It will be appreciated by those skilled in the art that in a practical single chip implementation as is disclosed herein, memory is an expensive resource in terms of chip area, and therefore must be minimized. Referring to Fig. 23, during Acquisition mode the FFT calculation process is not active so hardware can be shared between the FFT Window and the FFT calculation, most notably a 1024x22 RAM 232 used as a FIFO by the FFT Window, and selected for receipt of FFT data on line 234 by a multiplexer 236. Once in Tracking mode the FFT calculation process is active so that other control loops to recover sampling rate and frequency which depend on FFT data (e.g. pilots in the COFDM symbol) can initialize. Therefore tracking mode requires a dedicated tracking FIFO 238, which is selected by a multiplexer 240.

The input and output signals, and signals relating to the microprocessor interface 142 of the FFT Window circuitry shown in Fig. 23 are described in tables 15, 16, and 17 respectively.

In one embodiment a threshold level, set from statistical considerations, is applied to the F-ratio signal (see Fig. 7) to detect the negative and positive spikes which occur at the start and end of the guard interval respectively. The distance between the spikes is used to estimate the guard interval size. Repeated detection of the positive spikes is used to confirm correct synchronization. However with this method under noisy conditions the F-ratio signal becomes noisy and the spikes are not always reliably detectable.

In another embodiment peak detection is used to find the spikes in the F-ratios. It has been found that a fixed threshold is reliable only at or exceeding about a carrier-to-noise ("C/N") ratio of 12 dB. Peak detection is generally more sensitive and more specific, with generally reliable operation generally at 6 - 7 dB. The maxima should occur at the end of the guard interval. The difference in time between the two maxima is checked against the possible guard interval sizes. With an allowance for noise, the difference in time indicates the most likely guard interval size and the maxima themselves provide a good indication of the start of the active part of the symbol.

Preferably this process is iterated for several symbols to confirm detection, and is expected to improve performance when the C/N ratio is low.

The data stream is passed to accumulators 242, 244, each holding 64 moduli. Conversion to logarithms and subtraction of the logarithms is performed in block 246. The peaks are detected in peak detector block 248. Averaging of the symbol peaks is performed in block 250.

In noisy conditions, the maxima may be due to noise giving possibly inaccurate indications of the guard interval length and the start of the active symbol. The general strategy to cope with this is to perform a limited number of retries.

Currently, calculation of the F-ratio is done "on the fly" i.e. only once at each point. The variance estimates are calculated from 64 values only. Under noisy conditions, the variance estimates become very noisy and the spikes can become obscured. In an optional variation this problem is solved by obtaining more values for the variance estimate, by storing the variance estimate during acquisition for each of the possible $T+G_{max}$ points in the storage block 256. The variance estimates themselves may be formed by accumulating variances for each point, and then filtering in time over a number of symbols. A moving average filter or an infinite impulse response ("IIR") filter is suitable. A moving run of symbols, preferably between 16 and 32, are integrated in block 252, which increases the reliability of peak detection under noisy conditions. The storage block 256 holding the integrated F-ratio values is searched to find the maximum value. This is of length $T+G_{max}$, where G_{max} is the maximum guard interval size, T/4. Preferably the memory for storage block 256 is dynamically allocated, depending on whether acquisition mode or tracking mode is operative. Any unused memory is released to other processes. Similarly in tracking mode the integrated data stream is stored in tracking integration buffer 254.

This method has been tested with up to 4 symbols, without an IIR filter, and it has been found that the spikes can be recovered. However this approach does require increased memory.

FFT Processor

The discrete Fourier transform ("DFT") has the well known formula

$$x(k) = \frac{1}{L}\sum_{n=0}^{L-1} x(n)W^{nk} \qquad k = 0,1,\ldots,N-1 \qquad (25)$$

where N = the number of points in the DFT;

x(k) = the kth output in the frequency domain;

x(n) = the nth input in the time domain

and

$$W_L^{nk} = e^{-j(2\pi nk/L)} \qquad (26)$$

W is also known as a "twiddle factor".

For N > 1000 the DFT imposes a heavy computational burden and becomes impractical. Instead the continuous Fourier transform is used, given by

$$x(t) = \int_{t=-\infty}^{t=+\infty} x(t)e^{-j\omega t}dt \qquad (27)$$

The continuous Fourier transform, when computed according to the well known FFT algorithm, breaks the original N-point sequence into two shorter sequences. In the present invention the FFT is implemented using the basic butterfly unit 258 as shown in Fig. 24. The outputs C and D represent equations of the form C = A + B, and D = (A - B)W^k. The butterfly unit 258 exploits the fact that the powers of W are really just complex additions or subtractions.

A real-time FFT processor, realized as the FFT calculation circuitry 168 (Fig. 14) is a key component in the implementation of the multicarrier digital receiver 126 (Fig. 12). Known 8K pipeline FFT chips have been implemented with 1.5M transistors, requiring an area of 100 mm^2 in 0.5μ technology, based on the architecture of Bi and Jones. Even using a memory implementation with 3-transistor digital delay line techniques, over 1M transistors are needed. This has been further reduced with alternative architecture to 0.6M, as reported in the document *A New Approach to Pipeline FFT Processor*. Shousheng He and Mats Torkelson, Teracom Svensk RundRadio. DTTV-SA 180, TM 1547. This document proposes a hardware-oriented radix-2^2 algorithm, having radix-4 multiplicative complexity. However the requirements of the FFT computation in the present invention require the implementation of a radix 2^2+2 FFT processor.

Referring to Fig. 25 and Fig. 26 the butterfly structures BF2I 260 and BF2II 262, known from the noted Torkelson publication, are shown. The butterfly structure BF2II 262 differs from the butterfly structure BF2I 260 in that it has logic 264 and has a crossover 266 for crossing the real and imaginary inputs to facilitate multiplication by -j.

Fig. 27 illustrates the retimed architecture of a radix 2^2 + 2 FFT processor 268 in accordance with the invention, which is fully pipelined, and comprises a plurality of stages, stage-0 270 through stage-6 272. Except for stage-0 270, the stages each comprise one butterfly structure BF2I 260 and one butterfly structure BF2II 262, and storage RAMS 274, 276 associated therewith. stage-0 270 only has a single butterfly structure BF2I 260. This architecture performs a straight-forward 32-point FFT. stage-6 272 has control logic associated therewith, including demultiplexer 278 and multiplexer 280, allowing stage-6 272 to be bypassed, thus providing a 2K implementation of the FFT. Counters 282 configure the butterfly structures BF2I 260 and BF2II 262 to select one of the two possible diagonal computations, during which data is being simultaneously written to and read from the storage RAMS 274, 276.

Fig. 28 illustrates a 32 point flow graph of the FFT processor 268 using radix 2^2+2 pipeline architecture. Computations are performed using eight 4-point FFTs and four 8-point FFTs. These are decomposed in turn into two 4-point FFTs and four 2-point FFTs.

Fig. 29 illustrates the retimed architecture of a configurable 2K/8K radix 2^2+2 single path, delay feedback pipelined FFT processor 284, in which like elements in Fig. 27 are given the same reference numerals. The stages have a plurality of pipeline registers 286 which are required for proper timing of the butterfly structures BF2I 260 and BF2II 262 in the various stages. As can be seen, the addition of each pipelined stage multiplies the range of the FFT by a factor of 4. There are 6 complex multipliers 288, 290, 292, 294, 296, 298 which operate in parallel. This processor computes one pair of I/Q data points every four fast clock cycles, which is equivalent to the sample rate clock. Using 0.35μm technology the worst case throughput is 140μs for the 2K mode of operation, and 550μs for the 8K mode, exceeding the requirements of the ETS 300 744 telecommunications standard. Data enters the pipeline from the left side of Fig. 29, and emerges on the right. The intermediate storage requirements are 2K/8K for I data and 2K/8K for Q data, and is mode dependent. In practice the radix-4 stage is implemented as a cascade of two adapted radix-2 stages that exploit the radix-4 algorithms to reduce the number of required complex multipliers.

Fig. 30 is a schematic of one embodiment of the multipliers 288, 290, 292, 294, 296, 298 for performing the complex multiplication C = A x B, where A is data, and B is a coefficient. Because the FFT processor 284 has 6 complex multipliers, each requiring 3 hardware multipliers 300, a total of 18 hardware multipliers 300 would be required. It is preferable to use the embodiment of Fig. 31 in which some of the hardware multipliers 300 are replaced by multiplexers 302, 304.

Turning again to Fig. 29 there are a plurality of RAMS 306, 308, 310, 312, 314, 316 which are preferably realized as ROMs and contain lookup tables containing complex coefficients comprising cosines for the multipliers 288, 290, 292, 294, 296, 298 respectively. It has been discovered that by addressing the RAMS 306, 308, 310, 312, 314, 316 according to a particular addressing scheme, the size of these RAMS can be markedly reduced. The tradeoff between the complexity of the addressing circuitry and the reduction in RAM size becomes favorable beginning at stage-3 318. Referring again to Fig. 28 there are two columns 320, 322. Column 320 holds values W^2 - W^{14}, followed by W^1 - W^7, and then W^3 - W^{21}. These coefficients are stored in the RAM 308, required by the particular multiplier 290. Column 322 contains values W^8, W^4, W^{12}, which repeat 3 times. Note further that between the values W^8, W^4, and W^4, W^{12} are connections 324, 326 to the preceding butterfly unit located in column 328. In practice the connections 324, 326 are implemented as multiplications by W^0. In moving from multiplier to

multiplier toward the left in Fig. 29, the lookup table space is multiplied by a power of 4 at each stage. In Fig. 32 table 330, the lookup table for multiplier M^3 contains 512 entries. It can be deduced by extrapolation that multiplier M^5 must contain 8192 twiddle factors, and corresponds to the size of the FFT being performed by the FFT processor 284 (Fig. 29).

Before examining the look-up table space in more detail it is helpful to consider the plurality of horizontal lines 332. Moving downward from the top of Fig. 28, the line beginning at x(3) extends to W^8, which is the first twiddle factor required, and is at the third effective step in the flow diagram. Figs. 33 and 32 show the organization of the twiddle factors for each of the multipliers, wherein the terminology M_k represents the multiplier associated with the kth stage. Thus table 334 relates to multiplier M_0. The notation for the W values (twiddle factors) is shown in box 336. The subscript "B" at the bottom right represents a time stamp, that is an order dependency in which the twiddle factors are required by the pipeline. The superscript "A" represents the address of the twiddle factor in its lookup table. The superscript "N" is the index of the twiddle factor.

Thus in table 334 it may be seen that W^0 is required at time 0, W^1 at time 1, and W^0 is again required at time 2. Further inspection of the other tables in Figs. 33, 32 reveals that half of the entries in each table are redundant. The storage requirement for the lookup tables can be decreased by 50% by eliminating redundant entries. This has been accomplished by organizing the W values in ascending order by index, so that the values can be stored in memory in ascending order. Thus in the case of table 338 the index values range from 0 to 21, with gaps at 11, 13, 16, 17, 19, and 20.

The procedure for organizing the lookup table and the addressing scheme for accessing the twiddle factors is explained with reference to table 338, but is applicable to the other tables in Fig. 33. (1) Each row is assigned a line number as illustrated. (2) Each twiddle factor is assigned an order dependency which is noted in the lower right of its respective cell in table 338. (3) It is assumed that table 338 in its reduced form will contain only unique twiddle factors in ascending order by index within the memory address space. Consequently each twiddle factor is assigned a memory address as shown in the upper left of its respective cell.

During address generation, for line 3 of table 338 the address is simply held at 0. For line 1 the address is incremented by 1 to the end of the line. However lines 0 and 2 contain non-trivial address sequences. For line 0, looking at table 340, which contains 64 values, it will be observed that the address sequence changes according to the intervals 2,2,2,2, and then later 1,1,2,1,1,2 ... For line 2, the address first increments by 3, then by 2, and finally by 1. The locations at which the address increments change are

referred to herein as the "break-points". These values of the break points range between 0, corresponding to the first point in line 2, to the last position in the line.

By inspection it can be seen that the occurrence of the first break point changes from table to table following the recurrence relationship

$$B1_{M_N} = 4B1_{M_{N-1}} \qquad (28)$$

with the initial condition

$$B1_{M_0} = 1 \qquad (29)$$

where M_N is the multiplier of the Nth stage of the FFT processor 284.

Expanding the recurrence relationship gives:

$$B1_{M_N} = (((4B1_{M_0} - 1)x4 - 1)x4 - 1)\ldots \qquad (30)$$

$$B1_{M_N} = 4^N B1_{M_0} - 4^{N-3} - 4^{N-2} \ldots - 4^0 \qquad (31)$$

$$B1_{M_N} = 4^N B1_{M_N} - \sum_{n=0}^{N-1} 4^n \qquad (32)$$

Similarly the second break point B2 for line 2 is determined from the recurrence relation

$$B2_{M_N} = 4B2_{M_{N-1}} + 1 \qquad (33)$$

with the initial condition

$$B2_{M_0} = 1 \qquad (34)$$

or

$$B2_{M_N} = (((4B2_{M_0} + 1)x4 + 1)x4 + 1)\ \ldots \qquad (35)$$

$$B2_{M_N} = \sum_{n=0}^{N} 4^n \qquad (36)$$

Break point B3 for line 0 at which the sequence changes from increments of 2,2,2,2 to the pattern 1,1,2,1,1,2 . . . can be located by inspecting tables 338, 340, and 330. In table 338 the break point B3 occurs very late in the line, such that the second sequence only presents its first two elements. By examining the address locations in the larger noted tables, it can be deduced that the location of break point B3 is related to the number of entries in a particular table as

$$B3 = \frac{K}{4} + 2 \qquad (37)$$

where K is the number of table entries. In the tables in Fig. 29 K = 8, 32, 128, 2048, 8192. Therefore, in terms of the N'th complex multiplier, break point B3 can be expressed as

$$B3_{M_N} = 2 \times 4^N + 2 \qquad (38)$$

where $N \geq 0$.

Address generators 342, 344, 346, 348 are operative for the lookup tables in RAMS 310, 312, 314, 316. Silicon area savings for the smaller tables 308, 306 are too small to make this scheme worthwhile.

Fig. 34 schematically illustrates an address generator 342 for the above described address generation scheme, and is specific for the table 340 and multiplier M_2. 128 possible input states are accepted in lines in_Addr 350, and a multiplexer 352 selects the two most significant bits to decode 1 of 4 values. The output of the multiplexer 352 relates to the line number of the input state. Actually the output is the address increment applicable to the line number of the input state, and is used to control a counter 354 whose incremental address changes according to value on line 356. Thus, the increment for line 3 of table 340 is provided to the multiplexer 352 on line 358, and has a value of zero, as was explained above. Similarly the increment for line 1 of table 340 is provided to the multiplexer 352 on line 360, and has a value of 1.

The situations of line 0 and line 2 are more complicated. For line 0 the output of decoding logic 362 is provided by multiplexer 364, and has either an incremental value of 2, or the output of multiplexer 366. The latter could be either 1 or 2, depending on the state of a two bit counter 368, which feeds back a value of 0 or 1 as signal count 370.

Decoding logic 372 decodes the states for line 2 of table 340. The relationship of the current input state to the two break points of line 2 are tested by comparators 374, 376. The break point is actually set one sample earlier than the comparator output to allow for retiming. The outputs of the comparators 374, 376 are selectors for the multiplexers 378, 380 respectively.

The current address, held in accumulator 382 is incremented by the output of the multiplexer 352 by the adder 384. A simple logic circuit 386 resets the outgoing address, which is contained in register ACC 388, by asserting the signal rst 390 upon completion of each line of table 340. This insures that at the start of the next line the address points to twiddle factor W^0. The new address is output on the 6 bit bus out_Address 392, which is one bit smaller than the input in_Addr 350.

Fig. 35 is a generalization of address generator 342 (Fig. 34), in which the incoming address has a path of B bits. Like elements in Figs. 34 and 35 are given the same reference numerals. The structure of address generator 394 is similar to that of the address generator 342, except now the various lines of the input in_addr 396 and the output out_addr[B-2:0] 398 are denoted in terms of B. Thus the multiplexer 352 in Fig. 35 is selected by input in_addr [B-1:B-2] 400 . Similarly one of the inputs of comparator 374 and of comparator 376 is in_addr [B-3:0] 402. Out_addr[B-2:0] 398 forms the output. The advantage of this structure is a reduction in the size of the lookup table RAM of 50%.

The FFT calculation circuitry 168 (Fig. 14) is disclosed in Verilog code listings 1 - 17. The Verilog code for the address generator 394 is generic, enabling any power-of-four table to be implemented.

Channel Estimation and Correction

The function of the Channel estimation and correction circuitry shown in channel estimation and correction block 170 (Fig. 14) is to estimate the frequency response of the channel based on the received values of the continuous and scattered pilots specified in the ETS 300 744 telecommunications standard, and generate compensation coefficients which correct for the channel effects and thus reconstruct the transmitted spectrum. A more detailed block diagram of the channel estimation and correction block 170 is shown in Fig. 16.

In acquisition mode, the channel estimation and correction block 170 needs to locate the pilots before any channel estimation can take place. The circuitry performs a convolution across the 2048 carriers to locate the positions of the scattered pilots, which are always evenly spaced, 12 carriers apart. Having found the scattered pilots, the continual pilots can be located; once this is done the exact position of the 1705 active carriers within the 2048 outputs of the FFT calculation circuitry 168 (Fig. 14) is known. A timing generator 404 within the block can then be initialized, which then generates reference timing pulses to locate pilots for channel estimation calculation and for use in other functions of the demodulator as well.

Channel estimation is performed by using the evenly spaced scattered pilots, and then interpolating between them to generate the frequency response of the channel. The received carriers (pilots and data) are complex divided by the interpolated channel response to produced a corrected spectrum. A complete symbol is held in a buffer 406. This corrects for the bit-reversed order of the data received from the FFT calculation circuitry 168. It should be noted that raw, uncorrected data is required by the frequency and sampling rate error circuitry.

The task of synchronizing to the OFDM symbol in the frequency domain data received from the FFT calculation circuitry 168 (Fig. 14) begins with the localization of the scattered and continual pilots, which occurs in pilot locate block 408. Scattered pilots, which according to the ETS 300 744 telecommunications standard, occur every 12 data samples, offset by 3 samples with respect to the start of the frame in each succeeding frame. As the power of the pilot carriers is 4/3 the maximum power of any data carrier, a succession of correlations are performed using sets of carriers spaced at intervals of 12. One of the 12 possible sets is correlates highly with the boosted pilot carrier power.

A first embodiment of the pilot search procedure is now disclosed with reference to Figs. 36 and 16. It should be noted that the scattered pilot search procedure is done on the fly, and storage is only required in so far as is necessary to perform the subsequent step of continual pilot location discussed below. At step 410, after the assertion of the signal resync 204, generally occurring after a channel change or on power up, the signal pilot_lock 412 is set low. Then, at step 414 the process awaits the first symbol pulse from the FFT calculation circuitry 168 (Fig. 14) on line 416 indicating the start of the first symbol. The first symbol is received and stored. In one embodiment of the pilot search procedure each point from 0 to 2047 is read in turn, accumulating each value (|I| + |Q|) in one of 12 accumulators (not shown). The accumulators are selected in turn in a cycle of 12, thus convolving possible scattered pilot positions. Two well known peak trackers indicate the accumulator with highest value (Peak1) and the accumulator having the second highest value (Peak2). The accumulator having the highest value corresponds to the scattered pilot orientation. The second highest value is tracked so that the difference between the highest peak and the second highest peak can be used as a "quality" measure. At decision step 418, if the two peaks are not far enough apart, a test for completion of a full range frequency sweep is made at decision step 420. If the test fails, failure of the scattered pilot search is reported at step 422. Otherwise, at step 424 the IQ Demodulator LO frequency is incremented by +1/8 carrier spacing by incrementing the magnitude of the control signal freq_sweep 426. Then the search for scattered pilots is repeated after delaying 3 symbols at step 428 to allow time for the effect of the change to propagate through the FFT calculation circuitry 168 and buffers. The peak difference threshold can be altered by the control microprocessor via the microprocessor interface 142 and block 430.

In a variation of the first embodiment there is only a single peak tracker which indicates the accumulator with highest value, which corresponds to the scattered pilot orientation. The true scattered pilot orientation thus found is one of 12 possible orientations.

If the test at decision step 418 is successful, the search for continual pilots is begun at step 432 by establishing an initial pilot offset from the 0 location in the RAM, storing the FFT data, according to the formula

$$\text{pilot offset} = (\text{accumulator \# mod } 3) \quad \textbf{(39)}$$

Thus, if the scattered pilot peak is in accumulator 0, 3, 6 or 9 the pilot offset is 0. If the scattered pilot peak is in accumulator 1, 4, 7, or 10 then pilot offset is 1, etc. Then 45 carrier positions expected for continual pilots are read, adding the pilot offset value to the address, and accumulating (|I| + |q|) values. This procedure is repeated until first 115 continual pilot start positions have been searched. From the ETS 300 744 telecommunications standard the number of possible first carrier positions among the active carriers lying in a contiguous block between carrier 0 and carrier 2047 is easily calculated as (2048-1705) / 3 ≈115, as explained below. It is thus guaranteed that the active interval begins within the first (2048-1705) carrier positions. The carrier corresponding to the peak value stored is the first active carrier in the symbol.

Upon completion of the continual pilot search, at step 434 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase. The signal pilot_lock 412 is then set high at step 436, indicating that the pilots have been located successfully, then at step 436 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase.

In a tracking mode of operation, shown as step 438, the scattered pilot search is repeated periodically, and evaluated at decision step 440. This can be done at each symbol, or less frequently, depending upon propagation conditions. The predicted movement of the scattered pilot correlation peak is reflected by appropriate timing in the timing generator 404, and can be used as a test that timing has remained synchronized. Failure of the test at decision step 440 is reported at step 442, and the signal pilot_lock 412 is set low.

A second embodiment of the pilot search procedure is now disclosed with reference to Figs. 16 and 37. At step 444 the assertion of the signal resync 204, generally occurring after a channel change or on power up, the signal pilot_lock 412 is set low. Then, at step 446 a symbol is accepted for evaluation. A search for scattered pilots, conducted according to any of the procedures explained above, is performed at step 448. Then a search for continual pilots is performed as described above at step 450. At decision step 452 it is determined whether two symbols have been processed. If the test fails, control returns to step 446 and another symbol is processed. If the test succeeds at step 454 another test is made for consistency in the positions of the scattered and continual pilots in the two symbols. If the test at step 454 fails, then the

procedure beginning with decision step 420 is performed in the same manner as previously described with reference to Fig. 36. If the test at step 454 succeeds at step 456 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase. The signal pilot_lock 412 is then set high at step 458, indicating that the pilots have been located successfully.

In a tracking mode of operation, shown as step 460, the scattered pilot search is repeated periodically, and evaluated at decision step 462. This can be done at each cycle of operation, or less frequently, depending upon propagation conditions. The predicted movement of the scattered pilot correlation peak is reflected by appropriate timing in the timing generator 404, and can be used as a test that timing has remained synchronized. Failure of the test at decision step 462 is reported at step 464, and the signal pilot_lock 412 is set low.

It will be appreciated that after the scattered pilots have been located, the task of locating the continual pilots is simplified considerably. As the continual pilots are inserted at a known sequence of positions, the first of which is offset by a multiple of 3 positions with respect to start of the frame, as specified by the ETS 300 744 telecommunications standard. Two of three possible location sets in the data space can therefore be immediately excluded, and it is only necessary to search the third set. Accordingly the continual pilot search is repeated, each iteration beginning at a location 3 carriers higher. New accumulated values and the current start location are stored if they are larger than the previous accumulated value. This is repeated until all continual pilot start positions have been searched. The carrier corresponding to the largest peak value stored will be the first active carrier in the symbol. It is unnecessary to evaluate the "quality" of the continual pilot correlation peak. The scattered pilot search represents a correlation of 142 samples, and has higher noise immunity that of the search for 45 continual pilots. The continual pilot search is almost certain to be succeed if scattered pilot search completed successfully.

The above sequences locate scattered pilot positions within 1/4 symbol period, assuming accumulation at 40MHz, and locate continual pilots in less than 1 symbol period (45 x 115 clock cycles assuming 40MHz operation).

The I and Q data is provided to the pilot locate block 408 by the FFT calculation circuitry 168 (Fig. 14) in bit-reversed order on line 416. This complicates the problem of utilizing a minimum amount of RAM while computing the correlations during pilot localization. Incoming addresses are therefore bit reversed, and computed modulo 12 in order to determine which of 12 possible bins is to store the data. In order to avoid the square root function needed to approximate the carrier amplitude, the absolute values of the data are summed instead as a practical approximation. The scattered pilots are

determined "on the fly". The continual pilots are located on frames which succeed the frames in which the scattered pilots were located.

The operation of the timing generator 404 is now disclosed in further detail. The addressing sequence for the RAM buffer 406 is synchronized by a symbol pulse from the FFT calculation circuitry 168 (Fig. 14). The FFT calculation process runs continuously once the first symbol from has been received following FFT Window acquisition. Addressing alternates between bit-reversed and linear addressing for successive symbols. The timing generator 404 also generates all read-write timing pulses.

Signals u_symbol 466 and c_symbol 468 are symbol timing pulses indicating the start of a new uncorrected symbol or corrected symbol. The signal u_symbol 466 is delayed by latency of the interpolating filter 470 and the complex multiplier 472, which are synchronized to RAM Address Sequence Timing.

For carrier timing the signals c_carrier0 474, pilot timing signals us_pilot(+) 476, uc_pilot(+) 478, c_tps_pilot(*) 480 and odd_symbol pulse 482 are referenced to a common start pulse sequence. A base timing counter (not shown) is synchronized by the pilot locate sync timing pulse 484, and is therefore offset from symbol timing. Pilot timing outputs are also synchronized to uncorrected symbol output from the buffer 406 or the corrected symbol output delayed by the interpolating filter 470 and the complex multiplier 472. On assertion of the signal resync 204 all timing output is set to inactive states until the first symbol is received. Let the transmitted pilot at carrier k be P_k and the received pilot be P'_k.

$$P_k' = H_k \cdot w_k \cdot P_k \qquad (40)$$

where P_k is described below, and

$$P'_k = I_k + jQ_k \qquad (41)$$

where k indexes pilot carriers, H_k is the channel response and w_k is the reference sequence. We interpolate H_k to generate compensation values for the received data carriers, D'_k:

$$D_k' = I_k + jQ_k \qquad (42)$$

$$D_k = \frac{D_k'}{H_k} \qquad (43)$$

where k indexes data carriers. Received pilots can be demodulated using a locally generated reference sequence and are then passed to the interpolating filter.

The interpolating filter 470, realized in this embodiment with 6 taps and 12 coefficients, is utilized to estimate the portion of the channel between the scattered pilots. As explained above pilots are transmitted at known power levels relative to the data carriers and are modulated by a known reference sequence according to the ETS 300 744 telecommunications standard. The transmitted pilot carrier amplitudes are ± 4/3 of nominal data carrier power (+4/3 for reference bit of 1, -4/3 for the reference bit of 0; quadrature component = 0 in both cases). Interpolation coefficients are selected from the 0-11 cyclic count in the timing generator 404 synchronized to data availability. Appropriate correction factors may be selected for data points to provide on-the-fly correction. The coefficients vary depending on scattered pilot phase. Since the positions of reference pilots vary, therefore coefficients to compensate a given data carrier also vary.

The input and output signals, and signals relating to the microprocessor interface 142 of the channel estimation and correction block 170 are described in tables 18, 19 and 20 respectively. The circuitry of the channel estimation and correction block 170 is disclosed in Verilog code listings 18 and 19.

TPS Sequence Extract

The tps sequence extract block 172 (Fig. 14), although set out as a separate block for clarity of presentation, is in actuality partially included in the channel estimation and correction block 170. It recovers the 68-bit TPS data carried in a 68-symbol OFDM frame, and is shown in further detail in Fig. 38. Each bit is repeated on 17 differential binary phase shift keyed ("DBPSK") modulated carriers, the tps pilots, within a COFDM symbol to provide a highly robust transport channel. The 68-bit tps sequence includes 14 parity bits generated by a BCH code, which is specified in the ETS 300 744 telecommunications standard. Of course appropriate modifications can be made by those skilled in the art for other standards having different BCH encoding, and for modes other than 2K mode.

A clipper 486 clips incoming corrected spectrum data to ±1. The sign bit can be optionally evaluated to obtain the clipped result. In comparison block 488 clipped received tps pilot symbols are compared against a reference sequence input. In the described embodiment a value of 0 in the reference sequence matches -1 in the pilot, and a value of 1 in the reference sequence matches +1 in the pilot. Majority vote comparisons are used to provide an overall +1 or -1 result . A result of +1 implies the same modulation as the reference sequence, and a result of -1 implies inverse modulation.

The DBPSK demodulator 490 converts the +/-1 sequence from the majority vote form to a binary form. The sequence converts to a value of 0 if the modulation in current

and previous symbols was the same, and to 1 if modulation between successive symbols is inverted.

From an uninitialized condition a search for either of two sync words in 68-bit tps sequence (4 x 68-bit = 1 superframe) is conducted in the frame synchronizer block 492. The synchronization words of a superframe are as follows:

0011010111101110	sync word for frames 1 and 3
1100101000010001	sync word for frames 2 and 4

Having acquired either sync word, a search for the other is conducted in the appropriate position in the next OFDM frame. On finding the second sync word synchronization is declared by raising the signal tps_sync 494. Data is then passed to the BCH decoder 496, which operates on 14 parity bits at the end of an OFDM frame against received data in the frame. Errors are corrected as necessary.

Decoded data is provided to output store block 498, which stores tps data that is found in a full OFDM frame. The output store block 498 is updated only at the end of an OFDM frame. Only 30 bits of interest are made available. Presently some of these bits are reserved for future use. The length indicator is not retained.

The BCH decoder 496 has been implemented in a manner that avoids the necessity of performing the Berlekamp Algorithm and Chien Search which are conventional in BCH decoding. The Galois Field Multiplier used in the BCH decoder 496 is an improvement of the Galois Field Multiplier which is disclosed in our copending U.S. Application No. 08/801,544.

The particular BCH code protecting the tps sequence is specified in the ETS 300 744 telecommunications standard as BCH (67,53,t=2), having a code generator polynomial

$$h(x) = x^{14} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1 \qquad (44)$$

or equivalently

$$h(x) = (x^7 + x^3 + 1)\ (x^7 + x^3 + x^2 + x + 1) \qquad (45)$$

The left factor is used to generate the Galois Field which is needed for error detection. Referring to Fig. 39, this is calculated in syndrome calculation block 500 which can be implemented using a conventional feedback shift register to generate the α values. The first three syndromes are then computed by dividing the received signal R(x) by the values α^1, α^2, and α^3, again using a conventional feedback shift register implementation, as is well known in the art of BCH decoding. It can be shown that the syndromes are

$$S_0 = (\alpha^1)^{e_0} + (\alpha^1)^{e_1} \qquad (46)$$

$$S_1 = (\alpha^2)^{e_0} + (\alpha^2)^{e_1} \qquad (47)$$

$$S_2 = (\alpha^3)^{e_0} + (\alpha^3)^{e_1} \qquad (48)$$

During the syndrome computation the syndromes are stored in storage registers R[2:0] 502.

In the event S_0 is 0, then it can be immediately concluded that there are no errors in the current tps sequence, and a signal is asserted on line 504 which is provided to error detect block 506, and the data of the received signal R(x) either output unchanged or toggled according to the output of the error detect block 506 on line 508. As explained below, if

$$S_1 \odot S_0 = S_2 \qquad (49)$$

then exactly one error is present, a condition which is communicated to the error detect block 506 on line 510. Otherwise it is assumed that two errors are present. More than two errors cannot be detected in the present implementation.

In order to solve the system of three non-linear equations shown above, data flow from the registers R[2:0] 502 into search block 512 is enabled by a signal EOF 514, indicating the end of a frame. Three feedback shift registers 516, 518, 520 having respective Galois Field multipliers 522, 524, 526 for α^{-1} - α^{-3} in the feedback loop are initialized to 50H, 20H, and 3dH (wherein the notation "H" refers to hexadecimal numbers). The feedback shift registers 516, 518, 520 are clocked each time a new data bit is available. The syndromes and outputs of the feedback shift registers 516, 518, 520 are clocked into to a search module, which performs a search for the error positions using an iterative substitution search technique, which will now be described. The outputs of feedback shift registers 516, 518 are multiplied in a Galois Field Multiplier 528.

Considering the case of one error, S_0 is added, modulo 2, preferably using a network of XOR gates 530, to the output of the first feedback shift register 516 (α-gen$_0$). If the relationship

$$(S_0 + \alpha_{gen_0}) = 0 \qquad (50)$$

holds, it is concluded that there is an error in the present data bit. The bit being currently output from the frame store is toggled. The search is halted, and the data is output from the frame store.

Considering the case of two errors, if the following relationship holds, there is an error in the current bit being output from the frame store:

$$(S_0 + \alpha_{gen_0}) \odot (S_1 + \alpha_{gen_1}) = (S_2 + \alpha_{gen_2}) \quad \mathbf{(51)}$$

It is now necessary to store the three terms calculated in the immediately preceding equation into the registers R[2:0] 502 which previously stored the syndromes S_0 - S_2. This is represented by line 532.

The process continues, now looking for the second error, and reusing the data in registers R[2:0] 502, which now contains the syndromes as adjusted by the previous iteration. The adjusted syndromes are denoted S_0' - S_2'.

$$S_0' = (S_0 + \alpha_{gen_0}) \qquad ,\text{etc.} \qquad \mathbf{(52)}$$

Now, if

$$(S_0' + \alpha_{gen_0}) = 0 \qquad \mathbf{(53)}$$

the second error has been found, and the bit being currently output from the frame store is toggled by XOR gate 534. If the search fails, more than two errors may be present and an error signal (not shown) is set.

the Galois Field Multiplier 528 is a clocked digital circuit and is disclosed with reference to Fig. 40. The tps data is received very slowly, relative to the other processes occurring in the multicarrier digital receiver 126. It is thus possible to execute the iterative substitution search slowly, and the Galois Field Multipliers are designed for minimum space utilization. They do not require alpha generators, but rely on small constant coefficient multipliers, with iterative feedback to produce the required alpha values. The arrangement takes advantage of the relationship in Galois Field arithmetic

$$\alpha^n = \alpha^1 \cdot \alpha^{n-1} \qquad \mathbf{(54)}$$

After initialization by a signal init 536 which selects multiplexers 538, 540, the multiplicand A 542 is accumulated in register 544 and repeatedly multiplied by the value α^1 in multiplier 546. The output on line 548 is repeatedly ANDed bitwise with the multiplicand B held in a shift register 550. The output of the shift register is provided on a one bit line 552 to the gate 554. The output of the gate 554 is accumulated in register 556 using the adder 558.

The input and output signals and signals relating to the microprocessor interface 142 of the tps sequence extract block 172 are described in tables 21, 22, and 23. Circuitry of the tps sequence extract block 172 and the BCH decoder 496 is disclosed in Verilog code listings 20 and 21.

Automatic Fine Frequency Control and Automatic Sampling Rate Control

A non ideal oscillator present in the transmission chain of an orthogonal frequency division multiplexed ("OFDM") signal affects all carriers in the OFDM symbols. The OFDM carriers adopt the same phase and frequency disturbances resulting from the noisy local oscillator. Variations in the frequency of the Local Oscillator lead to phase shifts, and consequent loss of orthogonality within the OFDM symbol. Therefore competent automatic frequency control is required in the receiver to track the frequency offsets relative to the transmitter in order to minimize these phase shifts and hence maintain orthogonality.

All the carriers within an OFDM symbol are equally affected by the phase shifts. This is similar to the common phase error caused by phase noise. The common phase error present on all carriers is used to generate an Automatic Frequency Control ("AFC") signal, which is completely in the digital domain, since I/Q demodulation is performed in the digital domain. The approach taken is the calculation of the common phase error for every OFDM symbol. This is achieved by using the reference pilots. The change in the common phase error is measured over time to detect a frequency offset and is used to derive the AFC control signal. The generic approach for the AFC control loop and the automatic sampling rate control loop disclosed below is illustrated in Fig. 41.

Automatic sampling rate control is required when the receiver's master clock is not aligned with that of the transmitter. The misalignment causes two problems: (1) the demodulating carriers have incorrect spacing; and (2) the interval of the FFT calculation is also wrong.

The effect of this timing error is to introduce a phase slope onto the demodulated OFDM data. This phase slope is proportional to the timing error. The phase slope can be determined by calculating the phase difference between successive OFDM symbols, using reference pilots, and estimating the slope of these phase differences. A least squares approach is used for line fitting. The ASC signal is low-pass filtered and fed back to the sinc interpolator 158 (Fig. 13).

The mean phase difference between the reference pilots in subsequent OFDM symbols is used to calculate the frequency deviation. Assuming that the frequency deviations of the local oscillator are constant, then the phase rotates with α, where $\alpha = 2\pi f_d m T_t$ rads. Here f_d is frequency deviation, m is the number of symbols between repetitions of identical pilot positions, and T_t is the period comprising the sum of the active interval and the guard interval. The AFC signal is generated over time by low pass filtering α. The value of the frequency deviation is then used to control the IQ demodulator 144 (Fig. 13).

The AFC and ASC control signals are effective only when a guard interval is passing indicated by the assertion of signal IQGI on line 154 (Fig. 13). This prevents a symbol from being processed under two different conditions.

The correction circuitry 174 (Fig. 14) is shown in greater detail in Fig. 42. Frequency error values output on line 560 are calculated by determining the average of the differences of phase values of corresponding pilots in a current symbol and the previous symbol. The resulting frequency error value is filtered in low pass filter 562 before being fed-back to the IQ demodulator 144 (Fig. 13). It is optional to also evaluate continual pilots in order to cope with larger frequency errors. Sampling rate error, output on line 564 is determined by looking at the phase difference between pilots in a symbol and the same pilots in a previous symbol. The differences vary across the symbol, giving a number of points through which a line can be fitted using the well known method of least squares regression. The slope of this line is indicative of the magnitude and direction of the sampling rate error. The sampling rate error derived in this way is filtered in low pass filter 566 before being fed back to the sinc interpolator 158 (Fig. 13).

A separate store 568 for the scattered pilots contained in 4 symbols is shared by the frequency error section 570 and the sampling rate error section 572. Direct comparison of scattered pilot symbols is thereby facilitated, since the scattered pilot phase repeats every four symbols. In an alternate embodiment where scattered pilots are used to provide control information, storage must be provided for four symbols. In the preferred embodiment, wherein control information is derived from continual pilots, storage for only one symbol is needed.

Recovery of the angle of rotation α from the I and Q data is accomplished in the phase extract block 574, where

$$\alpha = \tan^{-1}(Q/I) \qquad (55)$$

In the presently preferred embodiment, the computations are done at a resolution of 14 bits. The phase extract block 574 is illustrated in greater detail in Fig. 43. The quadrant of α is first determined in block 576. The special cases where I or Q have a zero magnitude or I = Q is dealt with by the assertion of signals on lines 578. If the magnitude of Q exceeds that of I, quotient inversion is accomplished in block 580, utilizing a control signal 582. A positive integer division operation is performed in division block 584. Although this operation requires 11 clock cycles, there is more than enough time allocated for phase extraction to afford it. The calculation of the arctangent of the quotient is accomplished by a pipelined, truncated iterative calculation in block 586of the Taylor Series

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots, \qquad |x| < 1 \qquad (56)$$

Block 586 is shown in greater detail in the schematic of Fig. 44. The value x^2 is calculated once in block 588 and stored for use in subsequent iterations. Powers of x are then iteratively computed using feedback line 590 and a multiplier 592. The divisions are calculated using a constant multiplier 594 in which the coefficients are hardwired. The sum is accumulated using adder/subtractor 596. The entire computation requires 47 - 48 clock cycles at 40 MHz.

Turning again to Fig. 43, quadrant mapping, and the output of special cases is handled in block 598 under control of block 576. It may be noted that the square error of the result of the Taylor Expansion rises rapidly as α approaches 45 degrees, as shown in Fig. 45 and Fig. 46, which are plots of the square error at different values of α of the Taylor expansion to 32 and 31 terms respectively. The Taylor expansions to 31 and 32 terms are averaged, with the result that the square error drops dramatically, as shown in Fig. 47. A memory (not shown) for holding intermediate values for the averaging calculation is provided in block 598.

Constant Phase Error across all scattered Pilots is due to frequency offset at IQ Demodulator. Frequency Error can be defined as:

$$f_{err} = \frac{\alpha}{2\pi m T_t} \qquad (57)$$

where α, m and T_t have the same meanings as given above. α is determined by taking the average of the difference of phase values of corresponding pilots between the current symbol and a symbol delayed for m symbol periods. In the above equation, m = 1 in the case of continual pilots. This computation uses accumulation block 600 which accumulates the sum of the current symbol minus the symbol that preceded it by 4. Accumulation block 602 has an x multiplier, wherein x varies from 1 to a minimum of 142 (in 2K mode according to the ETS 300 744 telecommunications standard). The low pass filters 562, 566 can be implemented as moving average filters having 10 - 20 taps. The data available from the accumulation block 602 is the accumulated total of pilot phases each sampled m symbols apart. The frequency error can be calculated from

$$f_{err} = \frac{Acc\{new - old\}}{(N)(2)\pi m T_t} \qquad (58)$$

N = 142 in the case of scattered pilots, and 45 for continual pilots, assuming 2K mode of operation according to the ETS 300 744 telecommunications standard. The

technique for determining sampling rate error is illustrated in Fig. 48, in which the phase differences of pilot carriers, computed from differences of every fourth symbol (S_n - S_{n-4}) are plotted against frequency of the carriers. The line of best fit 604 is indicated. A slope of 0 would indicate no sampling rate error.

Upon receipt of control signal 606 from the pilot locate block 408 (Fig. 14), a frequency sweep is initiated by block 608, which inserts an offset into the low-pass filtered frequency error output using adder 610. Similarly a frequency sweep is initiated by block 612, which inserts an offset into the low-pass filtered sampling rate error output using adder 614. The frequency sweeps are linear in increments of 1/8 of the carrier spacing steps, from 0 - 3.5kHz corresponding to control signal values of 0x0-0x7.

A preferred embodiment of the correction circuitry 174 (Fig. 14) is shown in greater detail in Fig. 49. Continual pilots rather than scattered pilots are held in a memory store 616 at a resolution of 14 bits. The generation of the multiplier x for the computation in the accumulation block 618 is more complicated, since in accordance with the noted ETS 300 744 telecommunications standard, the continual pilots are not evenly spaced as are the scattered pilots. However, it is now only necessary to evaluate 45 continual pilots (in 2K mode according to the ETS 300 744 telecommunications standard). In this embodiment only the continual pilots of one symbol need be stored in the store 616. Inclusion of the guard interval size, is necessary to calculate the total duration of the symbol T_t, is received from the FFT window circuitry (block 166, Fig. 14) on line 620.

The input and output signals and signals relating to the microprocessor interface 142 of the circuitry illustrated in Fig. 42 are described in tables 24, 25, 26, and Table 27 respectively. The circuitry is further disclosed in Verilog code listings 24 - 35.

Demapper

The demapping circuitry 176 (Fig. 15) is shown as a separate block for clarity, but in practice is integrated into the channel estimation and correction circuitry. It converts I and Q data, each at 12-bit resolution into a demapped 12-bit coded constellation format (3-bit I, I soft-bit, 3-bit Q, Q soft-bit). The coded constellation is illustrated in Fig. 50 and Fig. 51. For 64-QAM the 3 bits are used for the I and Q values, 2 bits for 16-QAM 2-bits and 1 bit for QPSK.

For example in Fig. 51 values of I= 6.2, Q= -3.7 would be demapped to: I-data = 001; I soft-bit=011; Q-data=101; Q soft-bit=101.

The input and output signals of the demapping circuitry 176 are described in tables 28 and 29 respectively.

Symbol Deinterleaver

The symbol deinterleaver 182 (Fig. 15) reverses the process of symbol interleaving of the transmitted signal. As shown in Fig. 52 the deinterleaver requires a 1512 x 13

memory store, indicated as block 622. The address generator 624 generates addresses to write in interleaved data and read out data in linear sequence. In practice the address generator 624 is realized as a read address generator and a separate write address generator. Reading and writing occur at different instantaneous rates in order to reduce the burstiness of the data flow. The address generator 624 is resynchronized for each new COFDM symbol by a symbol timing pulse 626. Carrier of index 0 is marked by carrier0 pulse 628. Addresses should be generated relative to the address in which this carrier is stored.

The input and output signals of the symbol deinterleaver 182 are described in tables 30 and 31 respectively. Circuitry of the symbol deinterleaver 182 is disclosed in Verilog code listing 22.

Bit Deinterleaver

Referring to Fig. 54, the bit deinterleaver 184 (Fig. 15) reverses the process of bit-wise interleaving of the transmitted signal, and is shown further detail in Fig. 53. In soft encoding circuitry 630 input data is reformatted from the coded constellation format to a 24 bit soft I/Q format. The soft encoding circuitry 630 is disclosed for clarity with the bit deinterleaver 184, but is realized as part of the symbol deinterleaver discussed above. The deinterleave address generator 632 generates addresses to read the 6 appropriate soft-bits from the 126 x 24 memory store 634, following the address algorithm in the ETS 300 744 telecommunications standard. The deinterleave address generator 632 is resynchronized for each new COFDM symbol by the symbol timing pulse 626.

The output interface 636 assembles I and Q output data streams from soft-bits read from the memory store 634. Three I soft bits and three Q soft bits are extracted from the memory store 634 at each deinterleave operation, and are parallel-serial converted to provide the input data stream to the Viterbi Decoder 186 (Fig. 15).

The input and output signals of the bit deinterleaver 184 are described in tables 32 and 33 respectively. Circuitry of the bit deinterleaver 184 is disclosed in Verilog code listing 23.

Host Microprocessor Interface

The function of the microprocessor interface 142 is to allow a host microprocessor to access control and status information within the multicarrier digital receiver 126 (Fig. 12). The microprocessor interface 142 is shown in greater detail in Fig. 55. A serial interface 638 and a parallel interface 640 are provided, the latter being primarily of value for testing and debugging. The serial interface 638 is of known type and is I2C compatible. The microprocessor interface 142 includes a maskable interrupt capability allowing the receiver to be configured to request processor intervention depending on

internal conditions. It should be noted, that the multicarrier digital receiver 126 does not depend on intervention of the microprocessor interface 142 for any part of its normal operation.

The use of interrupts from the point of view of the host processor is now described. "Event" is the term used to describe an on-chip condition that a user might want to observe. An event could indicate an error condition or it could be informative to user software. There are two single bit registers (not shown) are associated with each interrupt or event. These are the condition event register and the condition mask register.

The condition event register is a one bit read/write register whose value is set to one by a condition occurring within the circuit. The register is set to one even if the condition only existed transiently. The condition event register is then guaranteed to remain set to one until the user's software resets it, or the entire chip is reset. The condition event register is cleared to zero by writing the value one. Writing zero to the condition event register leaves the register unaltered. The condition event register must be set to zero by user software before another occurrence of the condition can be observed.

The condition mask register is a one bit read/write register which enables the generation of an interrupt request if the corresponding condition event register is set. If the condition event is already set when 1 is written to the condition mask register an interrupt request will be generated immediately. The value 1 enables interrupts. The condition mask register clears to zero on chip reset. Unless stated otherwise a block will stop operation after generating an interrupt request and will restart soon after either the condition event register or the condition mask register are cleared.

Event bits and mask bits are always grouped into corresponding bit positions in consecutive bytes in the register map. This allows interrupt service software to use the value read from the mask registers as a mask for the value in the event registers to identify which event generated the interrupt. There is a single global event bit that summarizes the event activity on the chip. The chip event register presents the OR of all the on-chip events that have 1 in their respective mask bit. A value of 1 in the chip mask bit allows the chip to generate interrupts. A value of 0 in the chip mask bit prevents any on-chip events from generating interrupt requests. Writing 1 or 0 to the chip event register has no effect. The chip event register only clears when all the events enabled by a 1 in their respective mask bits have been cleared.

The IRQ signal 642 is asserted if both the chip event bit and the chip event mask are set. The IRQ signal 642 is an active low, "open collector" output which requires an

off-chip pull-up resistor. When active the IRQ output is pulled down by an impedance of 100Ω or less. A pull-up resistor of approx. 4kΩ is suitable.

The input and output signals of the microprocessor interface 142 are described in tables 34 and 35 respectively.

System Controller

The system controller 198 (Fig. 15), which controls the operation of the multicarrier digital receiver 126 (Fig. 12), in particular channel acquisition and the handling of error conditions, is shown in further detail in Fig. 56.

Referring to the state diagram in Fig. 57, the channel acquisition sequence is driven by four timeouts.

(1) AGC acquisition timeout. 20 ms (80 symbols) are allowed for the AGC to bring up the signal level, shown in step 644. Then the FFT window is enabled to start acquisition search in block 646.

(2) Symbol acquisition timeout: 200 symbol periods, the maximum guard interval plus active symbol length, is allocated to acquire the FFT window in step 648. Another 35 symbol periods are allocated to pilot location in step 650. Approximately 50 ms are required to process 2K OFDM symbols. An option is provided to exit step 650 as soon as the pilots have been located to save acquisition time in non-extreme situations.

(3) Control Loop Settling timeout: A further 10 ms, representing approximately 40 symbols is allocated to allow the control loops to settle in step 652. An option is provided to exit step 652 and return to an initial step resync 654 if pilots have been lost if control loop settling timeout occurs.

(4) Viterbi synchronization timeout: In block 656 approximately 150 symbol periods are allocated for the worst case of tps synchronization, indicated by step 658 and approximately 100 symbol periods for the Viterbi Decoder 186 (Fig. 15) to synchronize to the transmitted puncture rate, shown as step 660. This is approximately 65 ms. In reasonable conditions it is unnecessary to wait this long. As soon as Viterbi synchronization is established, then transition to the system_lock state 662. It is possible to bypass the tps synchronization requirement by setting parameters (see table below) in the receiver parameters register and setting set_rx_parameters to 1.

If acquisition fails at any stage, the process automatically returns to step resync 654 for retry.

Having acquired lock, the system will remain in lock unless a Reed-Solomon overload event occurs, i.e. the number of Reed-Solomon packets with uncorrectable errors exceeds a predetermined value (the rso_limit value) in any 1 second period. If any of the 4 synchronizing state machines in the acquisition sequence, FFT window (step 648), pilot locate (step 650), tps synchronization (step 658) and Viterbi synchroni-

zation (step 660), lose synchronization once channel acquisition has occurred, no action will be taken until an event, rso_event, occurs and the step resync 654 is triggered automatically.

In poor signal conditions acquisition may be difficult, particularly the Viterbi synchronization. Therefore a bit is optionally provided in the microprocessor interface 142 (Fig. 12), which when set extends the timeouts by a factor of 4.

The input and output signals, and the microprocessor interface registers of the system controller 198 are described in tables 36, 37, 38, and 39 respectively.

Tables

Pin Name	I/O	Description
Tuner/ADC Interface		
SCLK	O	Sample clock for ADC
IDATA[9:0]	I	Input ADC data bus (10-bit)
AGC	O	Automatic Gain Control to tuner(Sigma-Delta output)
XTC[2:0]	O	External Tuner Control Outputs
MPEG-2 Transport Interface		
OUTDAT[7:0]	O	MPEG-2 Transport Stream Data bus
OUTCLK	O	MPEG Transport Stream Output Clock
SYNC	O	MPEG Transport Stream Sync pulse (1 per 188byte)
VALID	O	MPEG Transport Stream Valid data flag
ERROR	O	MPEG Transport Stream Errored data flag
Serial Host Microprocessor Interface		
SD	I/O	Serial Interface Data
SC	I	Serial Interface Clock
SDT	I/O	Serial Data Through
SCT	O	Serial Clock Through (40MHz clock out when DEBUG is high)
SADDR[2:0]	I	Serial Address Inputs (Hardwired external value) used as TSEL pins when DEBUG is high
Parallel Host Microprocessor Interface		

Pin Name	I/O	Description
MA[5:0]	I	Microprocessor Address Bus
MD[7:0]	I/O	Microprocessor Data Bus 2-bit/DEBUG data @40MHz
MWE	I	Microprocessor Write Enable
MCE	I	Microprocessor Chip Enable
NOTIRQ	O	Interrupt Request
JTAG Test Access Port		
TCK	I	JTAG Test Clock
TMS	I	JTAG Test Mode Select
TDI	I	JTAG Test Data In
TDO	O	JTAG Test Data Out
NTRST	I	JTAG TAP Controller Reset
Miscellaneous Pins		
NRESET	I	Asynchronous Reset
CLK40	I	40MHz Input Clock
TSTRI	I	Transport Stream Interface tristate control
TA (MA[6])	I	Test Address Bit - Snooper access (Bit 7 of up address bus)
DEBUG	I	Test Pin
TSEL [2:0]/SADDR[2:0]	I	Internal Test Inputs (mux out internal data onto MD[7:0]) 0 = normal upi, 1= fft input data (24-bit), 2 = fft output data (24-bit), 3 = channel correction output data (24-bit), 4 = fec input data.(2 x 3-bit softbit) all data clocked out @40MHz, 24-bit data in 4 bytes. Clock brought out on SCT pin, for convenience. Symbol timing/other synch. signals indicated with market bits in data.
TLOOP	I	Test Input

Table 4

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x00	**Event Reg.**			
	0	R/W/0	chip_event	OR of all events which are interrupt-enabled (unmasked)
	1	R/W/0	lock_failed_event	Set to 1 if channel acquisition sequence fails
	2	R/W/0	rs_overload_event	Set to 1 if Reed-Solomon Decoder exceeds set threshold within one 1 second period
0x01	**Mask Reg.**			
	0	R/W/0	chip_mask	Set to 1 to enable IRQ output
	1	R/W/0	lock_failed_mask	Set to 1 to enable interrupt on channel acquisition fail
	2	R/W/0	rs_overload_mask	Set to 1 to enable interrupt on RS error threshold exceeded
0x02	**Status Reg.**			
	0	R/0	system_locked	Set to 1 when system acquired channel successfully
	1	R/0	viterbi_sync	Set to 1 when Viterbi is synchronized
	2	R/0	tps_sync	Set to 1 when OFDM frame carrying TPS data has been synchronized to.
	3	R/0	pilot_loc	Set to 1 when pilots in COFDM symbol have been located and synchronized to
	4	R/0	fft_loc	Set to 1 when guard interval has been located and synchronized to.
	7:5	R/1	viterbi_rate	Received Viterbi Code rate
0x04-0x05	**Control Reg:**			
	0	R/W/0	change_channel	When set to 1, holds device in "Reset" state. Clearing this bit initiates channel change.

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	1	R/W/0	agc_invert	Invert AGC Signa-Delta output. Default setting means low output associated with reduced AGC gain.
	2	R/W/0	o_clk_phase	Set to 1 to invert phase of output clock. Default condition: output data changes on falling edge of output clock.
	3	R/W/0	set_rx_parameters	Set to 1 to take Receiver Parameter Data from Receiver Parameter Register. Default condition: settings taken from TPS data (longer channel acquisition time)
	4	R/W/0	extend_agc	Set to 1 to hold acquisition sequence in agc_acquire state
	5	R/W/0	extend_fs	Set to 1 to hold acquisition sequence in fs_acquire state
	6	R/W/0	extend_settle	Set to 1 to hold acquisition sequence in fs_settle state
	7	R/W/0	extend_sync	When set to 1 to hold acquisition sequence in vit_sync state
	10:8	R/W/0	xtc	External Tuner Control bits (external pins XTC[2:0])
	11	R/W/0	i2c_gate	12C "Gate" signal; setting this to 1 enables the isolation buffer between the "processor side12C" bus and the "Tuner side" 12C so the processor can access a Tuner through COFDM device. Setting to 0 closes the "gate" to prevent 12C bus noise affecting delicate RF.

Ad-dress (Hex)	Bit No.	Dir/Re-set	Register Name	Description
	12	R/W/ (TSTRI)	ts_tri	Transport Stream Tristate control - set to 1 to tristate MPEG TS interface (e.g. to mux a QPSK device to same MPEG demux). Power-on state of TS output controlled by external pin TSTRI.
	13	R/W/0)	fast_ber	Set to 1 to reduce BER counter, vit_ill_state counter and rso_counter, counter periods from 1 sec to 100ms.
	15	R/W/0	soft_reset	Software Reset - set to 1 to reset all blocks except upi. Set to 0 to release.
0x06-0x07	Receiver Parameter Register:			
	15:14	R/W/2	upi_constellation	Constellation Pattern for Demapper and Bit Deinterleaver (reset condition = 64-QAM)
	13:12	R/W/0	upi_guard	Guard Interval: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4
	11:9	R/W/0	upi_alpha	Hierarchical Tranmission Mode or "alpha value" (reset condition = non-hierarchical mode)
	7:5	R/W/0	upi_hp_rate	Viterbi Code Rate for HP stream - in non-hierarchical mode this is taken as the Viterbi Code Rate (reset condition = 1/2 rate code)
	4:2	R/W/0	upi_lp_rate	Viterbi Code Rate for LP stream (reset condition = 1/2 rate code)
	1:0	R/W/0	upi_tx_mode	Tranmission mode (00=2K, 01=8K, others reserved)
0x08	7:0	R/W/0	rso_limit	Errored packet per second limit (for rs_overload_event bit)

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x09	7:0	R/0	rso_count	Count of Uncorrectable Transport Packets per second (saturates at 255). Write to register to latch a stable count value which can then be read back
0x0a-0x0b	15:0	R/0	ber	BER (before RS) deduced from RS corrections in 1 second period - max correctable bit errors ~1.35M/sec for 7/8, 64-QAM, 1/32 GI (equivalent to 43.e-3 BER assuming useful bitrate of 31.67 e6). Only top 16 bits of 21 bit counter are visible - resolution of ~1e-6 depending on code-rate, constellation GI length. Write to register to latch a stable count value which can then be read back.
0x0c-0x0d	15:0	R/0	agc_level	AGC "Control Voltage" (msb's)
0x0e-0x0f	11:0	R/0	freq_error	IQ Demodulator Frequency Error (from feedback loop)
0x10-0x13	TPS Data (including future use bits)			
	1:0	R/0	tps_frame	Number of last received cmplete OFDM frame in superframe
	3:2	R/0	tps_constellation	Constellation Pattern from TPS data
	7:5	R/0	tps_alpha	Hierachical Transmission Information
	10:8	R/0	tps_hp_rate	Viterbi Code Rate of High-Priority stream (In non-hierarchical mode this is the code rate of the entire stream)
	13:11	R/0	tps_lp_rate	Viterbi Code Rate of Low-Priority stream
	15:14	R/0	tps_guard_int	Guard Interval

Ad-dress (Hex)	Bit No.	Dir/Re-set	Register Name	Description
	17:16	R/0	tps_tx_mode	Transmission Mode
	31:19	R/0	tps_future	Undefined bits allocated for future use
*** Debug Access ***				
0x20-0x21	15	R/W/0	agc_open	Set to 1 to break AGC control loop
	11:0	R/W/0	agc_twiddle	AGC twiddle factor
0x22-0x23		R/W/0	agc_loop_bw	AGC Control loops parameters
0x24-0x25	15	R/W/0	freq_open	Set to 1 to break freq control loop
	14	R/W/0	freq_nogi	Set to 1 to allow frequency update anytime, not just during Guard Interval
	11:0	R/W/0	freq_twiddle	IQ Demod twiddle factor
0x26-0x27			freq_loop_bw	Frequency Control Loop parameters
0x28-0x29	15	R/W/0	sample_open	Set to 1 to break sample control loop
	14	R/W/0	sample_nogi	Set to 1 to allow sample update anytime, not just during Guard Interval
	11:0	R/W/0	sample_twiddle	Sampling Rate Twiddle factor
0x2a-0x2b		R/W/0	sample_loop_bw	Sampling Rate Control Loop parameters
0x2c-0x2d	11:0	R/0	sampling_rate_err	Sampling Rate Error (from feedback loop)
0x30-0x31	15	R/W/0	lock_fft_window	Set to 1 to prevent fft_window moving in Tracking mode
	14	R/W/0	inc_fft_window	Write 1 to move fft_window position one sample period later (one-shot operation)
	13	R/W/0	dec_fft_window	Write 1 to move fft_window position one sample period earlier (one-shot operation)
	12:0	R/0	fft_window	FFT Window position

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7:0	R/W/0	fft_win_thresh	FFT Window Threshold
0x34- 0x35	15	R/W/0	set_carrier_0	Set to 1 to use carrier_0 value as setting
	11:0	R/W/0	carrier_0	Carrier 0 position; readback value detected by Pilot Locate algorithm or force a value by writing over it
0x36	7:0	R/W/	csi_thresh	Channel State Information threshold - the fraction of mean level below which data carriers are marked by a bad_carrier flag. Nominally 0.2 (for 2/3 code rate).
0x37				
0x38- 0x39	11:0	R/0	vit_ill_states	Viterbi Illegal State Rate (per second)Write to register to latch count which can then be read back
***** SNOOPERS ***** (External test address bit TA[6] = 1)				
0x40- 0x41	15:14 11:0	R/WR/ W	T,IQGIFreq_error[11:0]	IQ Demod Snooper (Note: bit 0 = lsb of highest addressed byte, 21)
0x44- 0x47	31:30 27:16 11:0	R/W R/W R/W	T, Valid Q-data[11:0] I-data[11:0]	Low-Pass Filter Snooper
0x48- 0x4d	47:46 43:32 31 27:16 11:0	R/W R/W R/W R/W R/W	T,SincGI Sample_err[11:0] Valid Q-data[11:0] I-data[11:0]	Resampler Snooper
0x50- 0x53	31:29 27:16 11:0	R/W R/W R/W	T, Valid,Resync Q-data[11:0] I-data[11:0]	FFT Snooper
0x54- 0x57	31:30 29:28 27:16 11:0	R/W R/W R/W R/W	T, Valid, Symbol,Resync Q-data[11:0] I-data[11:0]	Channel Estimation & Correction Snooper
0x58- 0x5b	31:30 29:28 27:16 11:0	R/W R/W R/W R/W	T, Resync u_symbol, uc_pilot Q-data[11:0] I-data[11:0]	Frequency & Sampling Error Snooper

Ad-dress (Hex)	Bit No.	Dir/Re-set	Register Name	Description
0x5c-0x5f	31:30 29:28 27:16 15 11:0	R/W R/W R/W R/W R/W	T, Resync c_symbol, tps_pil. Q-data[11:0] reference_seq I-data[11:0]	TPS Sequence Extract Snoopers
0x60-0x65	39 36:35 34:32 27:16 15:14 13 11:0	R/W R/W R/W R/W R/W R/W R/W	T constellation alpha Q-data[11:0] Valid, c_symbol c_carrier0 I-data[11:0]	Demap Snooper
0x68-0x6a	23:22 21:20 19 11:0	R/W R/W R/W R/W	T, valid symbol, carrier0 odd_symbol demap_data[11:0]	Symbol Deinterleave Snooper
0x6c-0x6e	23:21 20:19 18:16 11:0	R/W R/W R/W R/W	T, valid, symbol constellation alpha symdi_data[11:0]	Bit Deinterleaver Snooper
0x70-0x71	15:13 6:4 2:0	R/W R/W R/W	T, valid, resync Q-data[2:0] I-data[2:0]	Viterbi Snooper
0x72-0x73	15:14 13:12 7:0	R/W R/W R/W	T, valid, resync, eop vit_data[7:0]	Forney Deinterleaver Snooper
0x74-0x75	15:14 13:12 7:0	R/W R/W R/W	T, valid, resync, eop deint_data[7:0]	Reed Solomon Snooper
0x76-0x77	15:14 13:12 11:0 7:0	R/W R/W R/W R/W	T, valid, resync, eop error_val, error deint_data[7:0]	Output Interface Snooper
0x78-0x7b	31 30:20 19:18 17 16 14 13:8 6:5 4:3 2:0	R/W R/W R/W R/W R/W R/W R/W R/W R/W R/W	T tps_data[10:0] pkt_err, err_val vit_ill_state vit_ill_val rs_corr_val rs_correct[5:0] vit_sync, tps_sync pilot_loc, fft_loc vit_rate[2:0]	System Controller Snooper

Table 5

Signal	Description
clk	40MHz main clock
clk20M	20MHz sample clock (used as a "valid" signal to indicate when valid input samples are received)
data[9:0]	sampled data input from ADC
agc_resync	control input; held low on channel change - on transition to high AGC should reset itself and accumulate new control voltage for new channel.
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus
upaddr[2:0]	Internal Microprocessor Address Bus (only 2-bits required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel1	Internal Address decode output (high = valid for 0x0c-0x0d)
upsel2	Internal Address decode output (high = valid for 0x20-0x23)
te, tdin	Scan inputs

Table 6

Signal	Description
agc	Signal - Delta modulated output signal; when integrated by external RC it provides an analogue representation of the internal digital "control voltage" valueInterpolated output data
tdout	scan outputs

Table 7

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x0c-0x0d	15:0	R/0	agc_level	AGC "Control Voltage" (msb's)
0x20-0x21	15	R/W/0	agc_open	Set to 1 to break AGC control loop
	11:0	R/W/0	agc_twiddle	AGC twiddle factor
0x22-0x23		R/W/0	agc_loop_bw	AGC Control loops parameters

Table 8

Signal	Description
clk	40MHz main clock
nrst	Active-low synchronous reset
clk20M	20MHz sample clock (used as a "valid" signal to indicate when input data sample is valid)
sample[9:0]	input data sample from ADC. (AGC should ensure that this white-noise-like signal is scaled to full dynamic range)
freq_err[11:0]	Frequency Error input - 1Hz accurate tuning over +/-0.5 carrier spacing
IQGI	Valid pulse for enable frequency error signal. The effect of the frequency control loop is held off until a guard interval is passing through the IQ Demod block. (IQGI is generated by the FFT window and indicates when a guard interval is passing).
te, tdin	Scan test inputs

Table 9

Signal	Description
I-data[11:0]	I data-stream to be low-pass filtered (40 MHZ timing)
Q-data[11:0]	Q data-stream to be low-pass filtered (40 MHZ timing)
valid	Valid output data indicator; high if data is being output on this clock cycle (40 MHZ timing)
tdout	Scan test output

Table 10

Signal	Description
clk	40MHz clock (2x sample clock)
nrst	Active-low synchronous reset
valid_in	high-pulse indicating valid data from IQ-demodulator (40MHz timing)
i_data[11:0], q_data[11:0]	input data from IQ-demodulator (20Msps)
te, tdin	Scan test inputs

Table 11

Signal	Description
i_out[11:0], q_out[11:0]	Low-Pass filtered output data

Signal	Description
valid	Output pulse indicating valid data output (decimated to 10Msps)
tdout	Scan test output

Table 12

Signal	Description
clk40M	40MHz main clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from low-pass filter (decimated to 10Msps)
sr_err[11:0]	SamplingRate Error feedback fro Freq/Sampling Error block
SincGI	Valid pulse for Error signal; effect of Sampling Rate contol loop is held off until guard interval is passing through Sinc Interpolator. FFT Window block generates this signal at appropriate time.
te,tdin	Scan test signals

Table 13

Signal	Description
i_out[11:0], q_out[11:0]	Interpolated output data
valid	Output pulse indicating valid data output)
tdout	Scan test output

Table 14

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0]	input data from front-end (ignore quadrature data for this block)
resync	Control signal: forces Sync FSM back to acquisition mode when pulsed high
guard[1:0]	Expected guard interval; programmed by Host uP to aid fft window acquisition. 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4

Signal	Description
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus (bi-directional)
upaddr[0]	Internal uP address bus (only 1-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Address decode output to select FFT window block

Table 15

Signal	Description
FFT_Window	Timing output pulse; low for 2048 samples indicating the active interval
fft_lock	Output pulse indicating status of Sync FSM; 1 = Symbol acquired
rx_guard[1:0]	Received Guard Interval Size: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = ¼
IQGI	Timing pulse indicating when the guard interval should arrive at the IQ demodulator (Frequency Error only corrected in the Guard Interval)
SincGI	Timing pulse indicating when the guard interval should arrive at the Sinc Interpolator (Sampling Error only corrected in the Guard Interval)
sr_sweep[3:0]	Sampling Rate sweep output; 4-Bit output used by Frequency and Sampling Error block to generate Sampling Rate "ping-pong" sweep during FFT window acquisition.

Table 16

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x30-0x32	15	R/W/0	lock_fft_window	Set to 1 to prevent fft_window moving in Tracking mode
	14	R/W/0	inc_fft_window	Write 1 to move fft_window position one sample period later (one-shot operation)
	13	R/W/0	dec_fft_window	Write 1 to move fft_window position one sample period earlier (one-shot operation)
	12:0	R/0	fft_window	FFT Window position

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7:0	R/W/0		

Table 17

Signal	Description
clk40M	40MHz clock (2x sample clock)
nrst	Synchronous reset (active low)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from FFT
symbol	Symbol timing pulse from FFT; high for first valid data value of a new symbol
resync	Resynchronization input triggered on e.g. channel change. Pulsed high to indicate return to acquisition mode (wait for first symbol pulse after resync before beginning pilot search)
lupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[0]	Internal uP address bus (only 1-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Internal address decode output; high for addresses 0x032-0x033

Table 18

Signal	Description
ui_data[11:0], uq_data[11:0]	Uncorrected spectrum data, as read from RAM buffer (for Frequency/Sampling Error block)
u_symbol	Uncorrected symbol start; high for first carrier of the un-corrected symbol
us_pilot	high for any carrier which is a scattered pilot in the uncor-rected symbol
ci_data[11:0], cq_data[11:0]	Corrected spectrum data; as output from the complex multiplier
valid	high for valid corrected symbol - data carriers only
bad_carrier	high if interpolated channel response for the carrier is below pre-set fraction of the mean of carriers of previous symbol - viterbi will discard the data carried by this carrier

Signal	Description
c_symbol	high for the first carrier in the corrected symbol
c_carrier0	high for the first active carrier in the corrected symbol (a continual pilot corresponding to a carrier index value of 0)
c_tps_pilot	high for any carrier in the corrected symbol which is a TPS pilot
pilot_lock	output high if pilots successfully located at the end of pilot acquisition phase.
odd_symbol	high for symbol period if symbol is odd number in frame (as determined from scattered pilot phase)
c_reference_seq	Reference sequence output to TPS Sequence block
freq_sweep[2:0]	Frequency Sweep control; incrementing 3-bit count which increments IQ Demodulator LO offset in Frequency and Sampling block. Sweeps 0-0.875 carrier spacing offset in 0.125 carrier spacing steps

Table 19

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x32-0x33	15	R/W/0	set_carrier_0	Set to 1 to use carrier_0 value as setting
	11:0	R/W/0	carrier_0	Carrier 0 position
0x36	7:0	R/W/	csi_thresh	Channel State Information threshold - the fraction of mean level below which data carriers are marked by a bad_carrier flag. Nominally 0.2 (for 2/3 code rate). A value of 0 would turn CSI off for comparison testing.
0x37	7:0			

Table 20

Signal	Description
clk40M	40MHz clock (2x sample clock)
ci_data[11:0]	corrected pilot data from Channel Estimation and Correction (only need I data because corrected pilots should only insignificant Im component; - only need sign bit)
tps_pilot	high for single clock cycle when data input is a tps_pilot - use like a valid signal.
reference_seq	Reference Sequence PRBS input from Channel Estimation & Correction - ignore for non-tps_pilot values
c_symbol	timing pulse high for 1 clock cycle for first carrier in new symbol (whether or not that carrier is active)
lupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[1:0]	Internal uP address bus (only 2-bits required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Internal address decode output; high for addresses 0x10-0x13

Table 21

Signal	Description
tps_data [29:0]	Output tps data (held static for 1 OFDM frame): tps_data[1:0] = frame number tps_data[3:2] = constellation tps_data[6:4] = hierarchy tps_data[9:7] = code rate, HP stream tps_data[12:10] = code rate, LP stream tps_data[14:13] = guard interval tps_data[16:15] = transmission mode tps_data[29:17] = future use bits Note that parameters are transmitted for the next frame; outputs should be double-buffered so parameters appear at block outputs in the correct frame (used by Demapper and Symbol/Bit deinterleave blocks to decode incoming data)
tps_sync	Status output from Frame Sync FSM - set to 1 when FSM is sync'd i.e when 2 valid sync words have been received in expected postions AND correct TPS data is available at the block outputs.

Table 22

0x10-0x13	TPS Data (including future use bits)			
	1:0	R/0	tps_frame	Number of last received complete OFDM frame in superframe

0x10-0x13	TPS Data (including future use bits)			
	3:2	R/0	tps_constellation	Constellation Pattern from TPS data
	7:5	R/0	tps_alpha	Hierarchical Transmission Information
	10:8	R/0	tps_hp_rate	Viterbi Code Rate of High-Priority stream (In non-hierarchical mode this is the code rate of the entire stream)
	13:11	R/0	tps_lp_rate	Viterbi Code Rate of Low-Priority stream
	15:14	R/0	tps_guard_int	Guard Interval
	17:16	R/0	tps_tx_mode	Transmission Mode
	31:19	R/0	tps_future	Undefined bits allocated for future use

Table 23

Signal	Description
clk40M	40MHz clock (2x sample clock)
nrst	Active low reset
us_pilot	input data valid signal; high when a scattered pilot is output from the Channel Estimation & Correction block
guard[1:0]	Guard Interval from which symbol period Tt can be deduced:00 = 1/32 (Tt = 231us) , 01 = 1/16 (238us), 10 = 1/8 (252us) , 11 = 1/4 (280us)
ui_data[11:0], uq_data[11:0]	input data from Channel Estimation & Correction (Uncorrected spectrum)
u_symbol	Symbol timing pulse from Channel Estimation & Correction; high for first valid data value of a new symbol (uncorrected spectrum)
resync	Resynchronization input triggered on e.g. channel change. Pulsed high to indicate return to acquisition mode (wait for first symbol pulse after resync before beginning Pilot search)
sr_sweep[3:0]	Sampling Rate Sweep control from FFT Window block; 0 = 0Hz offset, 1=+500Hz, 2=-500Hz,3=+1000Hz, 4=-1000Hz,5=+1500Hz,6=-1500Hz,7=+2000Hz,8=-2000Hz
freq_sweep[3:0]	Frequency Sweep control from Channel Estimation & Correction block; represents number n range 0-7 frequency offset = nx500Hz

Signal	Description
Iupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[3:0]	Internal uP address bus (only 4-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel1	Internal address decode output; high for addresses 0x0e-0x0f
upsel2	Address decode for addresses in range 0x24-0x2d

Table 24

Signal	Description
frequency_error	frequecy error output (to IQ Demod)
sampling_rate_error	Sampling Rate Error output (to Sinc Interpolator)
freq_lock	status output; high if frequency error low
sample_lock	status output; high if sampling rate error low

Table 25

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x0e-0x0f	11:0	R/0	freq_error	IQ Demodulator Frequency Error (from feedback loop)

Table 26

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x24-0x25	15	R/W/0	freq_open	Set to 1 to break freq control loop
	14	R/W/0	freq_nogi	Set to 1 to allow frequency update anytime, not just during Guard Interval
	11:0	R/W/0	freq_twiddle	IQ Demod twiddle factor
0x26-0x27			freq_loop_bw	Frequency Control Loop parameters
0x28-0x29	15	R/W/0	sample_open	Set to 1 to break sample control loop

Address (Hex)	Bit No.	Dir/Re-set	Register Name	Description
	14	R/W/0	sample_nogi	Set to 1 to allow sample update anythime, not just during Guard Interval
	11:0	R/W/0	sample_twiddle	Sampling Rate Twiddle factor
0x2a-0x2b		R/W/0	sample_loop_bw	Sampling Rate Control Loop parameters
0x2c-0x2d	11:0	R/0	sampling_rate_err	Sampling Rate Error (from feedback loop)

Table 27

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from Channel Estimation & Correction.
bad_carrier_in	Carrier Status falg - set if carrier falls below acceptable level; indicates to viterbi that data from this carrier should be discarded from error correction calculations.
c_symbol	Timing synchronization signal - high for the first data sample in the corrected COFDM symbol.
constellation[1:0]	control signal which defines constellation: 00 = QPSK, 01 = 16-QAM, 10 = 64-QAM
alpha[2:0]	control signal defining hierarchical transmission parameter, alpha: 000 = non-hierarchical transmission, 001 = alpha value of 1, 010 = alpha value of 2, 011 = alpha value of 4 (Note the first release of the chip will not support hierarchical transmission)

Table 28

Signal	Description
out_data[11:0]	deinterleaved output data 6 I, 6 Q format
bad_carrier	bad_carrier flag carried through demap process unchanged.
valid	Valid output data indicator; high if data is being output on this clock cycle

Signal	Description
d_symbol	Symbol timing pulse re-timed to synchronize with out_data

Table 29

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
demap_data[11:0]	input data from Demapper. Data is in 6-bit I, 6-bit Q format (for 64_QAM)
bad_carrier_in	Carrier status signal - set if carrier falls below limits; indicates to viterbi that data should be ignored. Carried with data as extra bit through deinterleaver store.
symbol	Timing synchronization signal - high for the first data sample in a COFDM symbol. Used to resynchronize address generation
carrier0	Timing pulse - high for the first active carrier (corresponding to carrier index value of 0) in a symbol
odd_symbol	high if symbol is odd number in the frame (different interleaving pattern in odd and even symbols within 68-symbol frame)

Table 30

Signal	Description
out_data[11:0]	deinterleaved output data coded constellation format
bad_carrier	Bad carrier output having passed through deinterleave RAM.
valid	Valid output data indicator; high if data is being output on this clock cycle
d_symbol	Output timing synchronization signal - high for first data sample in de-interleaved COFDM symbol.

Table 31

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored. Valid "spread out" to smooth out data rate over whole symbol - average of 1 data valid every six 40MHz cycles. Effective data rate at viterbi input dropped to 20MHz
sdi_data[11:0]	input data from Symbol Deinterleaver. Data is in 6-bit I, 6-bit Q format (for 64_QAM)

Signal	Description
bad_carrier	Set to 1 if a carrier conveying the data fell below cceptable limits; indicates to Viterbi that this data should be ignored
symbol	Timing synchronization signal - high for the first data sample in a COFDM symbol. Used to resynchronize address generation
constellation[1:0]	Constellation Type indicator:10 = 64-QAM01 = 16-QAM00 = QPSK
alpha[2:0]	Hierarchical transmission control:000 = non-hierarchical, 001 = alpha value 1, 010 = alpha value 2, 011 = alpha value 4(Note: in this first version of the device only non-hierarchical mode is supported)

Table 32

Signal	Description
I-data[2:0]	I soft-bit to Viterbi
discard-I	flag bit drived from bad_carrier signal; viterbi will ignore this soft-bit if set.(bad-carrier is repeated per soft-bit because of interleaving)
Q-data[2:0]	Q soft-bit to Viterbi
discard-Q	flag-bit; VIterbi will ignore this soft-bit if set
valid	Valid output data indicator; high if data is being output on this clock cycle

Table 33

Signal	Description
MD[7:0] (bi-di)	Microprocessor Data bus (bi-directional)
MA[5:0]	Microprocessor Address Bus
MR/W	Microprocessor Read / Write control
SCL	Serial Interface Clock
SDA(bi-di)	Serial Interface Data I/O (bi-directional - same pin as MD[0])
SADDR[2:0]	Serial Interface Address
S/P	Serial/Parallel interface select

Table 34

Signal	Description
nupdata[7:0] (bi-di)	Internal processor data bus (inverted) (bi-directional)

Signal	Description
upaddr[5:0]	Internal address bus (decoded to provide individual selects for various register banks within functional blocks)
upgrstr	Internal read strobe
upgwstr	Internal write strobe
IRQ	Interrupt Request (Active low, open collector)

Table 35

Signal	Description
pad_clk40	Uncontrolled 40MHz clock from input pad
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus (bi-directional)
upaddr[3:0]	Internal Microprocessor Address Bus (only bits relevant to registers within System Control)
uprstr	Internal Microprocessor Read strobe
upwstr	Internal Microprocessor Write Strobe
upsel1	block select decoded from microprocessor interface (1 = access to this block enabled) valid for addresses 0x00-0x0b
upsel2	address decode for 0x38-0x39 range
tps_data[10:0]	TPS data received in OFDM frame (1:0 = tps_constellation; 4:2 = tps_alpha7:5 = tps_hp_rate10:8 = tps_lp_rate)(Don't bother with Guard Interval - these parameters only affect back end blocks)
rs_correct[5:0]	Count of bits corrected in each RS packet (accumulated over 1 second for BER value)
rs_corr_val	Valid pulse; high when rs_correct value is valid
pkt_err	Set to 1 to indicate RS packet is uncorrectable; has >64 bit errors or is corrupted in some other way.
err_val	Set to 1 to indicate when pkt_err signal is valid
vit_ill_state	Viterbi_illegal state pulse; (accumulate to give Viterbi illegal state count)
vit_ill_val	NOW NOT REQUIRED - Viterbi illegal state valid pulse
vit_sync	Status signal - 1 if Viterbi is synchronized
tps_sync	Status signal - 1 if TPS is synchronized
pilot_loc	Status signal - 1 if pilot location completed successfully (found_pilots))
fft_loc	Status signal - 1 if FFT window has located correctly
vit_rate[2:0]	Received Viterbi puncture rate.

Signal	Description
tck	JTAG test clock - used for control of clock in test mode
njreset	JTAG test reset - for clock control block
jshift	JTAG test register shift control - for clock control block
j_ctrl_in	JTAG test data input

Table 36

Signal	Description
clk40	Test-controlled main clock
clk20	Test-controlled sample clock (input to IQ Demod and AGC)
lupdata[7:0] (bi-di)	Internal processor data bus (bi-directional)
nirq	Active Low interrupt request bit (derived from chip_event)
constellation[1:0]	Internal address bus (decoded to provide individual selects for various register banks within functional blocks)
alpha[2:0]	Hierarchical mode information
hp_rate[2:0]	Viterbi code rate for High Priority channel (in non-hierarchical mode this is the code rate for the complete channel)
lp_rate[2:0]	Viterbi code rate for Low Priority channel.
upi_tx_mode[1:0]	Transmission mode (2K or 8K)
upi_guard[1:0]	Guard Interval
rxp_valid	Set to 1 if Host Interface has set rx_para data - used as a "valid" signal for rx_para data (in case of TPS data use tps_sync)
o_clk_phase	Control line; set to 1 to invert output clock phase
xtc[2:0]	External Tuner Control bits
i2c_gate	I2C "Gate" control
ts_tri	Transport Stream Interface tristate control
soft_reset	Software Reset (set to 1 to reset everything except upi)
agc_invert	Control line: set to 1 to invert sense of AGC sigma-delta output (default: low output equates to low AGC gain)
agc_resync	Control line: When set low AGC held in initial condition. Resync transitioning high commences the AGC acquisition sequence

Signal	Description
fft_resync	Control line: hold low to re-initialise FFT, Channel Estimation & Correction, Frequency/Sampling Error and TPS blocks. Transition high commences FFT window locate, Pilot locate and TPS synchronisation.
viterbi_resync	Contol line; hold low to re-initiliase FEC backend. Transition high commences Viterbi synchronisation.
j_ctrl_out	JTAG test data output - from clock control block.

Table 37

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x00	**Event Reg.**			
	0	R/W/0	chip_event	OR of all events which are interrupt-enabled (unmasked)
	1	R/W/0	lock_failed_event	Set to 1 if channel acquisition sequence fails
	2	R/W/0	rs_overload_event	Set to 1 if Reed-Solomon Decoder exceeds set threshold within one 1 second period
0x01	**Mask Reg.**			
	0	R/W/0	chip_mask	Set to 1 to enable IRQ output
	1	R/W/0	lock_failed_mask	Set to 1 to enable interrupt on channel acquisition fail
	2	R/W/0	rs_overload_mask	Set to 1 to enable interrupt on RS error threshold exceeded
0x02	**Status Reg.**			
	0	R/0	system_locked	Set to 1 when system acquired channel successfully
	1	R/0	viterbi_sync	Set to 1 when Viterbi is synchronized
	2	R/0	tps_sync	Set to 1 when OFDM frame carrying TPS data has been synchronized to.

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	3	R/0	pilot_loc	Set to 1 when pilots in COFDM symbol have been located and synchronized to
	4	R/0	fft_loc	Set to 1 when guard interval has been located and synchronized to.
	7:5	R/1	viterbi_rate	Received Viterbi Code rate
0x04-0x05	Control Reg:			
	0	R/W/0	change_channel	When set to 1, holds device in "Reset" state. Clearing this bit initiates channel change.
	1	R/W/0	agc_invert	Invert AGC Signa-Delta output. Default setting means low output associated with reduced AGC gain.
	2	R/W/0	o_clk_phase	Set to 1 to invert phase of output clock. Default condition: output data changes on falling edge of output clock.
	3	R/W/0	set_rx_parameters	Set to 1 to take Reciver Parameter Data from Receiver Parameter Register. Default condition: settings taken from TPS data (longer channel acquisition time)
	4	R/W/0	extend_agc	Set to 1 to hold acquisition sequence in agc_acquire state
	5	R/W/0	extend_fs	Set to 1 to hold acquisition sequence in fs_acquire state
	6	R/W/0	extend_settle	Set to 1 to hold acquisition sequence in fs_settle state

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7	R/W/0	extend_syn	When set to 1 to hold acquisition sequence in vit_sync state
	10:8	R/W/0	xtc	External Tuner Control bits (external pins XTC[2:0])
	11	R/W/0	i2c_gate	I2C "Gate" signal; setting this to 1 enables the isolation buffer between the "processor side" I2C bus and the "Tuner side" I2C so the processor can acces a Tuner through COFDM device. Setting to 0 closes the "gate" to prevent I2C bus noise affecting delicate RF.
	12	R/W/0	ts_tri	Transport Stream Tristate control - set to 1 to tristate MPEG TS interface (eg. to mux a QPSK devce to same MPEG demux). Power-on state of TS output controlled by external pin - somehow!!!
	13	R/W/0	fast_ber	Set to 1 to reduce BER counter, vit_ill_state counter and rso_counter, counter periods from 1 sec to 100ms
	15	R/W/0	soft_reset	Software Reset - set to 1 to reset all blocks except upi. Set to 0 to release.
0x06-0x07	Receiver Parameter Register:			
	15:14	R/W/2	upi_constellation	Constellation Pattern for Demapper and Bit Deinterleaver (reset condition = 64-QAM)
	13:12	R/W/0	upi_guard	Guard Interval: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	11:9	R/W/0	upi_alpha	Hierarchical Tranmission Mode or "alpha value" (reset condition = non-hierarchical mode)
	7:5	R/W/0	upi_hp_rate	Viterbi Code Rate for HP stream - in non-hierarchical mode this is taken as the Viterbi Code Rate (reset condition = 1/2 rate code)
	4:2	R/W/0	upi_lp_rate	Viterbi Code Rate for LP stream (reset condition = 1/2 rate code)
	1:0	R/W/0	upi_tx_mode	Trnnsmission mode (00=2K, 01=8K, others reserved)
0x08	7:0	R/W/0	rso_limit	Errored packet per second limit (for rs_overload_event bit)
0x09	7:0	R/0	rso_count	Count of Uncorrectable Transport Packets per second (saturates at 255).Write to register to latch a stable count value which can then be read back.
0x0a - 0x0b	15:0	R/0	ber	BER (before RS) deduced from RS corrections in 1 second period - max correctable bit errors ~1.35M/sec for 7/8, 64-QAM, 1/32 GI (equivalent to 43.e-3 BER assuming useful bitrate of 31.67 e6). Only top 16 bits of 21 bit counter are visible - resolution of ~1e-6 depending on code-rate, constellation GI length. Write to register to latch a stable count value which can then be read back.

Table 38

0x38-0x39	11:0	R/0	vit_ill_states	Viterbi Illegal State Rate (per second) Write to register to latch count which can then be read back

Table 39

Listing 1

```
// SccsId: %W% %G%
/*****************************************************************************
   Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for butterfly processor BF2I. (RTL)

 Notes   : Computes first stage in radix 4 calculation.

*****************************************************************************/

`timescale 1ns / 100ps

module fft_bf2I (clk, enable_1, in_x1r, in_x1i, in_x2r, in_x2i, in_s,
        out_z1r, out_z1i, out_z2r, out_z2i, out_ovf);

 parameter        wordlength = 5;     // Data wordlength.

 input           clk,           // Master clock.
              enable_1,        // Enable on clock 3.
                in_s;         // Control line.
 input [wordlength-1:0]  in_x1r,          // Input I from memory.
              in_x1i,        // Input Q from memory.
                in_x2r,         // Input I stage n-1.
                in_x2i;         // Input Q stage n-1.

 output          out_ovf;        // Overflow flag.
 output [wordlength-1:0]  out_z1r,         // Output I to stage n+1
              out_z1i,        // Output Q to stage n+1
                out_z2r,        // Output I to memory.
                out_z2i;        // Output Q to memory.

 wire  [wordlength-1:0]  in_x1r,
              in_x1i,
                in_x2r,
                in_x2i,
              out_z1r,
                out_z1i,
                out_z2r,
                out_z2i;
 wire           in_s,
              enable_1,
              out_ovf;

 reg  [wordlength-1:0]  z1r_tmp1,
              z1i_tmp1,
                z2r_tmp1,
                z2i_tmp1,
              z1r_tmp2,
                z1i_tmp2,
                z2r_tmp2,
```

```
             z2i_tmp2;
reg          ovf_tmp,
           ovf_tmp0,
           ovf_tmp1,
             ovf_tmp2,
             ovf_tmp3,
           ex_reg0,
             ex_reg1,
             ex_reg2,
             ex_reg3;

always @(in_s or in_x1r or in_x1i or in_x2r or in_x2i)
begin
 {ex_reg0,z1r_tmp1} = in_x1r + in_x2r;
 ovf_tmp0 = in_x1r[wordlength-1] &&           // Overflow check.
      in_x2r[wordlength-1] &&
       ~z1r_tmp1[wordlength-1] ||
      ~in_x1r[wordlength-1] &&
        ~in_x2r[wordlength-1] &&
        z1r_tmp1[wordlength-1];
 if (ovf_tmp0)                      // Saturate logic.
  z1r_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
         {1'b0,{wordlength-1{1'b1}}};

 {ex_reg1,z1i_tmp1} = in_x1i + in_x2i;
 ovf_tmp1 = in_x1i[wordlength-1] &&           // Overflow check.
      in_x2i[wordlength-1] &&
      ~z1i_tmp1[wordlength-1] ||
      ~in_x1i[wordlength-1] &&
        ~in_x2i[wordlength-1] &&
        z1i_tmp1[wordlength-1];
 if (ovf_tmp1)                      // Saturate logic.
  z1i_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
           {1'b0,{wordlength-1{1'b1}}};

 {ex_reg2,z2r_tmp1} = in_x1r - in_x2r;
 ovf_tmp2 = in_x1r[wordlength-1] &&           // Overflow check.
      ~in_x2r[wordlength-1] &&
        ~z2r_tmp1[wordlength-1] ||
      ~in_x1r[wordlength-1] &&
        in_x2r[wordlength-1] &&
      z2r_tmp1[wordlength-1];
 if (ovf_tmp2)                      // Saturate logic.
  z2r_tmp1 = (ex_reg2) ? {1'b1,{wordlength-1{1'b0}}} :
         {1'b0,{wordlength-1{1'b1}}};

 {ex_reg3,z2i_tmp1} = in_x1i - in_x2i;
 ovf_tmp3 = in_x1i[wordlength-1] &&           // Overflow check.
      ~in_x2i[wordlength-1] &&
      ~z2i_tmp1[wordlength-1] ||
      ~in_x1i[wordlength-1] &&
      in_x2i[wordlength-1] &&
        z2i_tmp1[wordlength-1];
 if (ovf_tmp3)                      // Saturate logic.
  z2i_tmp1 = (ex_reg3) ? {1'b1,{wordlength-1{1'b0}}} :
         {1'b0,{wordlength-1{1'b1}}};
```

```
// Output stage with two channel mux.
if (!in_s)
 begin: mux_passthru
   z1r_tmp2 = in_x1r;
   z1i_tmp2 = in_x1i;
   z2r_tmp2 = in_x2r;
   z2i_tmp2 = in_x2i;
 end
else
 begin: mux_computing
   z1r_tmp2 = z1r_tmp1;
   z1i_tmp2 = z1i_tmp1;
   z2r_tmp2 = z2r_tmp1;
   z2i_tmp2 = z2i_tmp1;
 end
end

assign out_z1r = z1r_tmp2;
assign out_z1i = z1i_tmp2;
assign out_z2r = z2r_tmp2;
assign out_z2i = z2i_tmp2;

always @(posedge clk)
 if (enable_1)     // Butterfly completes at the end of clock cycle 0.
  ovf_tmp <= in_s && (ovf_tmp0 || ovf_tmp1 || ovf_tmp2 || ovf_tmp3);

assign out_ovf = ovf_tmp;

`ifdef OVERFLOW_DEBUG_LOW_LEVEL
// Debug code to display overflow output of a particular adder.
// Concurrently monitor overflow flag and halt on overflow.
always @(ovf_tmp or ovf_tmp0 or ovf_tmp1 or ovf_tmp2 or ovf_tmp3)
 if (ovf_tmp)
  begin
    if (ovf_tmp0) $display("ovf_tmp0 on BF2I = ",ovf_tmp0);
    if (ovf_tmp1) $display("ovf_tmp1 on BF2I = ",ovf_tmp1);
    if (ovf_tmp2) $display("ovf_tmp2 on BF2I = ",ovf_tmp2);
    if (ovf_tmp3) $display("ovf_tmp3 on BF2I = ",ovf_tmp3);
    $stop;
  end
`endif
endmodule
```

Listing 2

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for butterfly processor BF2II. (RTL)

 Notes   : Computes second stage in radix 4 calculation.
```

```
*****************************************************************************/

`timescale 1ns / 100ps

module fft_bf2II (clk, enable_1, in_x1r, in_x1i, in_x2r, in_x2i, in_s, in_t,
        out_z1r, out_z1i, out_z2r, out_z2i, out_ovf);

 parameter      wordlength = 5;     // Data wordlength.

 input        clk,          // Master clock.
          enable_1,         // Enable on clock 3.
            in_s,         // Control line.
            in_t;         // Control line.
 input [wordlength-1:0] in_x1r,          // Input I from memory.
          in_x1i,         // Input Q from memory.
            in_x2r,         // Input I stage n-1.
            in_x2i;         // Input Q stage n-1.

 output       out_ovf;          // Overflow flag.
 output [wordlength-1:0] out_z1r,          // Output I to stage n+1
          out_z1i,         // Output Q to stage n+1
            out_z2r,         // Output I to memory.
            out_z2i;         // Output Q to memory.

 wire  [wordlength-1:0] in_x1r,
          in_x1i,
            in_x2r,
            in_x2i,
          out_z1r,
            out_z1i,
            out_z2r,
            out_z2i;
 wire         in_s,
          in_t,
            enable_1,
            out_ovf,
            control;

 reg  [wordlength-1:0] z1r_tmp1,
          z1i_tmp1,
            z2r_tmp1,
            z2i_tmp1,
          z1r_tmp2,
            z1i_tmp2,
            z2r_tmp2,
            z2i_tmp2,
          x2ri_tmp1,
            x2ri_tmp2;
 reg          ovf_tmp,
          ovf_tmp0,
          ovf_tmp1,
            ovf_tmp2,
            ovf_tmp3,
          ex_reg0,
            ex_reg1,
            ex_reg2,
```

```
              ex_reg3;

    assign control = in_s && !in_t;

    always @(in_s or control or in_x1r or in_x1i or in_x2r or in_x2i)
    begin
     // Crosspoint switch, used in computing complex j values.
     if (control)
      begin: switch_crossed
       x2ri_tmp1 = in_x2i; // i -> r.
       x2ri_tmp2 = in_x2r; // r -> i.
      end
     else
      begin: switch_thru
       x2ri_tmp1 = in_x2r; // r -> r.
       x2ri_tmp2 = in_x2i; // i -> i.
      end

     {ex_reg0,z1r_tmp1} = in_x1r + x2ri_tmp1;
     ovf_tmp0 = in_x1r[wordlength-1] &&           // Overflow check.
          x2ri_tmp1[wordlength-1] &&
            ~z1r_tmp1[wordlength-1] ||
          ~in_x1r[wordlength-1] &&
            ~x2ri_tmp1[wordlength-1] &&
            z1r_tmp1[wordlength-1];
     if (ovf_tmp0)                         // Saturate logic.
      z1r_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
               {1'b0,{wordlength-1{1'b1}}};

     {ex_reg1,z1i_tmp1} = (control) ? in_x1i - x2ri_tmp2:in_x1i + x2ri_tmp2;
     ovf_tmp1 = in_x1i[wordlength-1] &&           // Overflow check.
          (control ^ x2ri_tmp2[wordlength-1]) &&   // Deals with a
            ~z1i_tmp1[wordlength-1] ||          // +/- input.
          ~in_x1i[wordlength-1] &&
            ~(control ^ x2ri_tmp2[wordlength-1]) &&
            z1i_tmp1[wordlength-1];
     if (ovf_tmp1)                         // Saturate logic.
      z1i_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
               {1'b0,{wordlength-1{1'b1}}};

     {ex_reg2,z2r_tmp1} = in_x1r - x2ri_tmp1;
     ovf_tmp2 = in_x1r[wordlength-1] &&           // Overflow check.
            ~x2ri_tmp1[wordlength-1] &&         // Deals with a
            ~z2r_tmp1[wordlength-1] ||          // - input.
          ~in_x1r[wordlength-1] &&
            x2ri_tmp1[wordlength-1] &&
            z2r_tmp1[wordlength-1];
     if (ovf_tmp2)                         // Saturate logic.
      z2r_tmp1 = (ex_reg2) ? {1'b1,{wordlength-1{1'b0}}} :
               {1'b0,{wordlength-1{1'b1}}};

     {ex_reg3,z2i_tmp1} = (control) ? in_x1i + x2ri_tmp2:in_x1i - x2ri_tmp2;
     ovf_tmp3 = in_x1i[wordlength-1] &&           // Overflow check.
          ~(control ^ x2ri_tmp2[wordlength-1]) &&  // Deals with a
            ~z2i_tmp1[wordlength-1] ||          // -/+ input.
          ~in_x1i[wordlength-1] &&
```

```
        (control ^ x2ri_tmp2[wordlength-1]) &&
        z2i_tmp1[wordlength-1];
    if (ovf_tmp3)                // Saturate logic.
     z2i_tmp1 = (ex_reg3) ? {1'b1,{wordlength-1{1'b0}}} :
           {1'b0,{wordlength-1{1'b1}}};

    // Output stage with two channel mux.
    if (!in_s)
     begin: mux_passthru
       z1r_tmp2 = in_x1r;
       z1i_tmp2 = in_x1i;
       z2r_tmp2 = x2ri_tmp1;
       z2i_tmp2 = x2ri_tmp2;
     end
    else
     begin: mux_computing
       z1r_tmp2 = z1r_tmp1;
       z1i_tmp2 = z1i_tmp1;
       z2r_tmp2 = z2r_tmp1;
       z2i_tmp2 = z2i_tmp1;
     end
    end

    assign out_z1r = z1r_tmp2;
    assign out_z1i = z1i_tmp2;
    assign out_z2r = z2r_tmp2;
    assign out_z2i = z2i_tmp2;


    always @(posedge clk)
     if (enable_1)      // Butterfly completes at the end of clock cycle 0.
      ovf_tmp <= in_s && (ovf_tmp0 || ovf_tmp1 || ovf_tmp2 || ovf_tmp3);

    assign out_ovf = ovf_tmp;


    `ifdef OVERFLOW_DEBUG_LOW_LEVEL
    // Debug code to display overflow output of a particular adder.
    // Concurrently monitor overflow flag and halt on overflow.
    always @(ovf_tmp or ovf_tmp0 or ovf_tmp1 or ovf_tmp2 or ovf_tmp3)
     if (ovf_tmp)
      begin
        if (ovf_tmp0) $display("ovf_tmp0 on BF2II = ",ovf_tmp0);
        if (ovf_tmp1) $display("ovf_tmp1 on BF2II = ",ovf_tmp1);
        if (ovf_tmp2) $display("ovf_tmp2 on BF2II = ",ovf_tmp2);
        if (ovf_tmp3) $display("ovf_tmp3 on BF2II = ",ovf_tmp3);
        $stop;
      end
     `endif
    endmodule
```

Listing 3

```
// SccsId: %W% %G%
/**************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited
```

```
Author   : Dawood Alam.

Description: Verilog code for a variable size ROM with complex data store.
    (RTL)

Notes   : Used to store complex Twiddle factors.

*****************************************************************************/

`timescale 1ns / 100ps

module fft_rom (clk, enable_3, address, rom_data);

 parameter        c_wordlength = 1;    // Coeff wordlength.
 parameter        rom_AddressSize = 1;  // Address size.
 parameter FILE = "../../../fft/src/lookup_tables/lu_10bit_2048pt_scaleX";
                  // Lookup tab filename. (Listings 16, 17)

 input          clk,
             enable_3;
 input [rom_AddressSize-1:0] address;

 output [c_wordlength-1:0]  rom_data;

 reg  [c_wordlength*2-1:0] rom [0:(1 << rom_AddressSize)-1];
 reg  [c_wordlength*2-1:0] b_tmp1,
             rom_data;

 always @(address)
  b_tmp1 = rom[address];

 always @(posedge clk)
  if (enable_3)
   rom_data <= b_tmp1;

 initial
  $readmemb(FILE, rom);

endmodule
```

Listing 4

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for variable length single bit shift register.

 Notes   : Used to delay pipeline control signals by "length" clocks.

*****************************************************************************/

`timescale 1ns / 100ps
```

```
module fft_sr_1bit (clk, enable_3, in_data, out_data);

 parameter       length = 1;       // Shift reg length.

 input        clk,         // Master clock;
           enable_3;       // Enable on clock 3.
 input        in_data;       // Input data.

 output         out_data;        // Output data.

 reg          shift_reg [length-1:0];  // Shift register.

 wire         out_data;
 wire         clk,
           enable_3;
 integer        i;

 always @ (posedge clk)
  if (enable_3)
   begin
     for (i = (length-1); i >= 0; i = i - 1)
      if (i == 0)
       shift_reg[0] <= in_data;        // Force input to SR.
      else
       shift_reg[i] <= shift_reg[i-1];     // Shift data once.
   end

 assign out_data = shift_reg[length-1];
endmodule
```

Listing 5

```
// SccsId: %W% %G%
/******************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author  : Dawood Alam.

 Description: Verilog code for a dual-port FIFO. (RTL)

 Notes  : Used as a pipeline register to delay address into the address
     decoder.

******************************************************************************/

`timescale 1ns / 100ps

module fft_sr_addr (clk, enable_3, in_data, out_data);

 parameter       wordlength = 1;     // Data wordlength I/Q.
 parameter       length = 1;       // Shift reg length.

 input        clk,         // Master clock;
           enable_3;       // Enable on clock 3.
 input  [wordlength-1:0]  in_data;       // SR input data.
 output [wordlength-1:0]  out_data;        // SR output data.
```

```
reg  [wordlength-1:0]  shift_reg [length-1:0];  // Shift register.

wire  [wordlength-1:0]  out_data;
wire            clk,
             enable_3;
integer         i;

always @ (posedge clk)
 if (enable_3)
  begin
   for (i = (length-1); i >= 0; i = i - 1)
     if (i == 0)
      shift_reg[0] <= in_data;        // Force input to SR.
    else
      shift_reg[i] <= shift_reg[i-1];    // Shift data once.
  end

 assign out_data = shift_reg[length-1];
endmodule
```

Listing 6

```
// SccsId: %W% %G%
/* Copyright (c) 1997 Pioneer Digital Design Centre Ltd.

 Author   : Dawood Alam.

 Description: Verilog code for an signed twiddle factor multiplier. (RTL)

 Notes   : Single multiplexed multiplier and 2 adders employed to
      perform 3 multiplies and 5 additions. Pipeline depth = 2.
      ar/ai = Complex data, br/bi = Complex coefficient.
      bi +/- br could be pre-calculated in the ROM lookup, however
      in this implementation it is NOT an overhead as this path is
         shared by ar + ai. */

`timescale 1ns / 100ps

module fft_complex_mult_mux (clk, c2, in_ar, in_ai, in_br, in_bi,
                out_cr, out_ci, out_ovf);

 parameter      wordlength = 12;    // Data wordlength.
 parameter      c_wordlength = 10;   // Coeff wordlength.
 parameter      mult_scale = 4;     // multiplier scalling,
                // 1 = /4096, 2 = /2048,
                // 3 = /1024, 4 = /512.

 input [wordlength-1:0]  in_ar,        // Data input I.
           in_ai;     // Data input Q.
 input [c_wordlength-1:0] in_br,        // Coefficient input I.
           in_bi;     // Coefficient input Q.
 input        clk;       // Master clock.
 input [1:0]      c2;         // Two bit count line.
 output        out_ovf;      // Overflow flag.
 output [wordlength-1:0]  out_cr,        // Data output I.
           out_ci;     // Data output Q.
```

```
wire [wordlength-1:0] in_ar,
        in_ai,
          br_tmp,
          bi_tmp,
          out_cr,
          out_ci;
wire [c_wordlength-1:0] in_br,
        in_bi;
wire       enable_0,
          enable_1,
          enable_2,
          enable_3;
wire [1:0]     c2;

reg [wordlength-1:0] in_ai_tmp,
        in_ar_tmp,
          abr_tmp,
        abi_tmp,
          abri_tmp1,
          abri_tmp2,
          abri_tmp4,
          coeff_tmp1,
        mpy_tmp1,
          sum_tmp0,
          sum_tmp1,
          sum_tmp2,
          acc_tmp,
          store_tmp,
          cr_tmp,
          ci_tmp;
reg [wordlength*2-1:0] abri_tmp3,
        mpy_tmp2,
          coeff_tmp2;
reg       ovf_tmp0,
        ovf_tmp1,
          ovf_tmp2,
          ovf_tmp3,
        ex_reg0,
          ex_reg1,
          c1, c3, c4;

// Enable signals for registers.
assign enable_0 = ~c2[1] && ~c2[0];
assign enable_1 = ~c2[1] && c2[0];
assign enable_2 =  c2[1] && ~c2[0];
assign enable_3 =  c2[1] && c2[0];

// Sign extend coefficients from c_wordlength bits to wordlength.
assign br_tmp = {{(wordlength-c_wordlength){in_br[c_wordlength-1]}},in_br};
assign bi_tmp = {{(wordlength-c_wordlength){in_bi[c_wordlength-1]}},in_bi};

// Combinational logic before pipeline register.
always @(in_ar or br_tmp or in_ai or bi_tmp or c2)
 begin
```

```
c1 = c2[0] || c2[1];
c3 = c2[1];

if (!c1)
  begin
   abr_tmp = in_ar;
   abi_tmp = in_ai;
  end
else
  begin
   abr_tmp = br_tmp;
   abi_tmp = bi_tmp;
  end

if (c3)
  {ex_reg0,abri_tmp4} = abi_tmp - abr_tmp;
else
  {ex_reg0,abri_tmp4} = abi_tmp + abr_tmp;

ovf_tmp0 = abi_tmp[wordlength-1] &&          // Overflow check.
     (c3 ^ abr_tmp[wordlength-1]) &&        // Deals with a
     ~abri_tmp4[wordlength-1] ||            // +/- input.
     ~abi_tmp[wordlength-1] &&
     ~(c3 ^ abr_tmp[wordlength-1]) &&
     abri_tmp4[wordlength-1];

if (ovf_tmp0)                    // Saturate logic.
  abri_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
          {1'b0,{wordlength-1{1'b1}}};
else
  abri_tmp1 = abri_tmp4;

end

// Combinational logic after pipeline register.
always @(in_ar_tmp or in_ai_tmp or br_tmp or c2 or store_tmp or abri_tmp2)
 begin
  c4 = c2[1] && ~c2[0];

  case (c2)
  2'b00:
   begin
     coeff_tmp1 = in_ar_tmp;
     sum_tmp0 = store_tmp;
    end
  2'b01:
   begin
     coeff_tmp1 = br_tmp;
     sum_tmp0 = {wordlength-1{1'b0}};
    end
  2'b10:
   begin
     coeff_tmp1 = in_ai_tmp;
     sum_tmp0 = store_tmp;
    end
  2'b11:
```

```
  begin
    coeff_tmp1 = in_ar_tmp;
    sum_tmp0 = store_tmp;
   end
 endcase

 abri_tmp3 = {{wordlength{abri_tmp2[wordlength-1]}},abri_tmp2}; // extnd
 coeff_tmp2 = {{wordlength{coeff_tmp1[wordlength-1]}},coeff_tmp1};// extnd
 mpy_tmp2  = (abri_tmp3 * coeff_tmp2);

 mpy_tmp1 = mpy_tmp2[wordlength*2-mult_scale:wordlength-(mult_scale-1)];

 if (c4)
   {ex_reg1,sum_tmp2} = sum_tmp0 - mpy_tmp1 - mpy_tmp2[wordlength-mult_scale];
 else
   {ex_reg1,sum_tmp2}     =     mpy_tmp1     +     sum_tmp0     +
mpy_tmp2[wordlength-mult_scale];

 ovf_tmp1 = (c4 ^ mpy_tmp1[wordlength-1]) &&      // Overflow check.
     sum_tmp0[wordlength-1] &&          // Deals with a
     ~sum_tmp2[wordlength-1] ||         // +/- input.
     ~(c4 ^ mpy_tmp1[wordlength-1]) &&
     ~sum_tmp0[wordlength-1] &&
     sum_tmp2[wordlength-1];
 if (ovf_tmp1)                   // Saturate logic.
    sum_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
             {1'b0,{wordlength-1{1'b1}}};
 else
    sum_tmp1 = sum_tmp2;
 end

 // Pipeline registers for I/Q data paths and intermediate registers.
 always @(posedge clk)
  begin
    if (enable_2) // Enable on 2nd clock.
     acc_tmp <= sum_tmp1;             // Temp store.

    if (enable_3) // Enable on 3rd clock.
     cr_tmp <= acc_tmp;              // Pipeline reg cr

    if (enable_3) // Enable on 3rd clock.
     ci_tmp <= sum_tmp1;             // Pipeline reg ci

    if(enable_1)
    store_tmp <= sum_tmp1;            // Temp store.

    if (enable_2)
     in_ar_tmp <= in_ar;             // Reg i/p to mpy.

    if (enable_1)
     in_ai_tmp <= in_ai;             // Reg i/p to mpy.

    if (enable_0 || enable_1 || enable_2)
     abri_tmp2 <= abri_tmp1;           // Pipeline reg.

  end
```

```
// Register ovf outputs before final OR, else whole complex multiplier is
// treated as combinational, and the intermediate pipeline reg is ignored.
always @(posedge clk)
// if (enable_0 || enable_1 || enable_2)
  ovf_tmp2 <= ovf_tmp0;

always @(posedge clk)
 ovf_tmp3 <= ovf_tmp1;

assign out_ovf = ovf_tmp2 || ovf_tmp3;

`ifdef OVERFLOW_DEBUG_LOW_LEVEL
// Debug code to display overflow output of a particular adder.
// Concurrently monitor overflow flag and halt on overflow.
always @(posedge clk)
 if (out_ovf)
  begin
   if (ovf_tmp2) $display("ovf_tmp0 on complex multiplier = ",ovf_tmp2);
   if (ovf_tmp3) $display("ovf_tmp1 on complex multiplier = ",ovf_tmp3);
   $stop;
  end
`else
`endif

assign out_cr = cr_tmp;
assign out_ci = ci_tmp;

endmodule
```

Listing 7

```
// SccsId: %W% %G%
/******************************************************************************
   Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for a dual-port FIFO with complex data store. (RTL)

 Notes    : A variable bitwidth FIFO shift register for intermediate I/Q
     calculations.

******************************************************************************/

`timescale 1ns / 100ps

module fft_sr_iq (clk, enable_3, in_xr, in_xi, out_xr, out_xi);

 parameter       wordlength = 1;     // Data wordlength I/Q.
 parameter       length = 1;        // Shift reg length.

 input          clk,           // Master clock;
           enable_3;        // Enable on clock 3.
 input  [wordlength-1:0]  in_xr,         // SR input data, I.
           in_xi;         // SR input data, Q.
```

```
output [wordlength-1:0]  out_xr,          // SR output data I.
            out_xi;         // SR output data Q.

reg   [wordlength-1:0]  shift_r [length-1:0];   // SR for I data.
reg   [wordlength-1:0]  shift_i [length-1:0];   // SR for Q data/

wire  [wordlength-1:0]  out_xr,
            out_xi;
wire          clk,
            enable_3;
integer       i;

always @ (posedge clk)
 if (enable_3)
  begin
    for (i = (length-1); i >= 0; i = i - 1)
     begin
      if (i == 0)
       begin
    shift_r[0] <= in_xr;         // Force input I to SR.
        shift_i[0] <= in_xi;         // Force input Q to SR.
       end
      else
       begin
    shift_r[i] <= shift_r[i-1];      // Shift data I once.
        shift_i[i] <= shift_i[i-1];      // Shift data Q once.
       end
     end
  end

 assign out_xr = shift_r[length-1];
 assign out_xi = shift_i[length-1];
endmodule
```

Listing 8

```
// SccsId: %W% %G%
/**************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for 8 hardwired coefficients in a lookup table, of
      which 4 are unique values.

 Notes    : Used to store complex Twiddle factors. 8 point FFT twiddle factor
      coefficients (Radix 4+2). Coefficients stored as non-fractional
      10 bit integers. Real Coefficient (cosine value) is coefficient
      high-byte. Imaginary Coefficient (sine value) is coefficient
      low-byte. Coefficient addresses are delayed by a pipeline depth
      of 5, i.e. equivalent to case table values being advanced by 5.

 **************************************************************************/

`timescale 1ns / 100ps
```

```
module fft_hardwired_lu0 (clk, enable_3, address, out_br, out_bi);

 parameter        c_wordlength = 10;    // Coeff wordlength.
 parameter        rom_AddressSize = 3;   // Address bus size.

 input          clk,
            enable_3;
 input [rom_AddressSize-1:0] address;

 output [c_wordlength-1:0]  out_br, out_bi;

 reg  [c_wordlength*2-1:0] b_tmp1,
            b_tmp2;

 always @(address)
 case (address)
  3'd6: b_tmp1 = 20'b0000000000_1000000000; // W2_8 = +0.000000 -1.000000
  3'd0: b_tmp1 = 20'b0101101010_1010010110; // W1_8 = +0.707107 -0.707107
  3'd2: b_tmp1 = 20'b1010010110_1010010110; // W3_8 = -0.707107 -0.707107
  default:b_tmp1 = 20'b0111111111_0000000000;// W0_8 = +1.000000 -0.000000
 endcase

 always @(posedge clk)
  if (enable_3)
   b_tmp2 <= b_tmp1;

 assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
 assign out_bi = b_tmp2[c_wordlength-1:0];

 endmodule
```

Listing 9

```
// SccsId: %W% %G%
/******************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

  Author   : Dawood Alam.

  Description: Verilog code for 32 hardwired coefficients in a lookup table, of
      which 16 are unique values.

  Notes   : Used to store complex Twiddle factors. 32 point FFT twiddle
      factor coefficients (Radix 4+2). Coefficients stored as
      non-fractional 10 bit integers. Real Coefficient (cosine value)
      is coefficient high-byte. Imaginary Coefficient (sine value) is
      coefficient low-byte. Coefficient addresses are delayed by a
      pipeline depth of 4, i.e. equivalent to case table values being
      advanced by 4.

 ******************************************************************************/

 `timescale 1ns / 100ps

 module fft_hardwired_lu1 (clk, enable_3, address, out_br, out_bi);
```

```
parameter        c_wordlength = 10;    // Coeff wordlength.
parameter        rom_AddressSize = 5;  // Address bus size.

input            clk,
             enable_3;
input [rom_AddressSize-1:0] address;

output [c_wordlength-1:0]  out_br, out_bi;

reg  [c_wordlength*2-1:0] b_tmp1,
            b_tmp2;

always @(address)
case (address)
 5'd5,
 5'd14:b_tmp1 = 20'b0111011001_1100111100;// W02_32 = +0.923880 -0.382683
 5'd6,
 5'd16:b_tmp1 = 20'b0101101010_1010010110;// W04_32 = +0.707107 -0.707107
 5'd7,
 5'd18,
 5'd22:b_tmp1 = 20'b0011000100_1000100111;// W06_32 = +0.382683 -0.923880
 5'd8: b_tmp1 = 20'b0000000000_1000000000;// W08_32 = +0.000000 -1.000000
 5'd9: b_tmp1 = 20'b1100111100_1000100111;// W10_32 = -0.382683 -0.923880
 5'd10,
 5'd24:b_tmp1 = 20'b1010010110_1010010110;// W12_32 = -0.707107 -0.707107
 5'd11:b_tmp1 = 20'b1000100111_1100111100;// W14_32 = -0.923880 -0.382683
 5'd13:b_tmp1 = 20'b0111110110_1110011100;// W01_32 = +0.980785 -0.195090
 5'd15,
 5'd21:b_tmp1 = 20'b0110101010_1011100100;// W03_32 = +0.831470 -0.555570
 5'd17:b_tmp1 = 20'b0100011100_1001010110;// W05_32 = +0.555570 -0.831470
 5'd19:b_tmp1 = 20'b0001100100_1000001010;// W07_32 = +0.195090 -0.980785
 5'd23:b_tmp1 = 20'b1110011100_1000001010;// W09_32 = -0.195090 -0.980785
 5'd25:b_tmp1 = 20'b1000001010_1110011100;// W15_32 = -0.980785 -0.195090
 5'd26:b_tmp1 = 20'b1000100111_0011000100;// W18_32 = -0.923880 +0.382683
 5'd27:b_tmp1 = 20'b1011100100_0110101010;// W21_32 = -0.555570 +0.831470
 default: b_tmp1 = 20'b0111111111_0000000000;// W00_32 = +1.000000 -0.000000
 endcase

 always @(posedge clk)
  if (enable_3)
   b_tmp2 <= b_tmp1;

 assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
 assign out_bi = b_tmp2[c_wordlength-1:0];

endmodule
```

Listing 10

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.
```

Description: Verilog code for 128 hardwired coefficients in a lookup table, of which 64 are unique values.

Notes : Used to store complex Twiddle factors. 128 point FFT twiddle factor coefficients (Radix 4+2). Coefficients stored as non-fractional 10 bit integers. Real Coefficient (cosine value) is coefficient high-byte. Imaginary Coefficient (sine value) is coefficient low-byte. Coefficient addresses are delayed by a pipeline depth of 3, i.e. equivalent to case table values being advanced by 3.

```
*******************************************************************************/

`timescale 1ns / 100ps

module fft_hardwired_lu2 (clk, enable_3, address, out_br, out_bi);

 parameter       c_wordlength = 10;    // Coeff wordlength.
 parameter       rom_AddressSize = 7;   // Address bus size.

 input           clk,
            enable_3;
 input [rom_AddressSize-1:0] address;

 output [c_wordlength-1:0]  out_br, out_bi;

 reg  [c_wordlength*2-1:0] b_tmp1,
            b_tmp2;

 always @(address)
 case (address)
  7'd36:b_tmp1 =20'b0111111111_1111100111; //W01_128=+0.998795 -0.049068
  7'd4,
  7'd37:b_tmp1 =20'b0111111110_1111001110; //W02_128=+0.995185 -0.098017
  7'd38,
  7'd68:b_tmp1 =20'b0111111010_1110110101; //W03_128=+0.989177 -0.146730
  7'd5,
  7'd39:b_tmp1 =20'b0111110110_1110011100; //W04_128=+0.980785 -0.195090
  7'd40:b_tmp1 =20'b0111110001_1110000100; //W05_128=+0.970031 -0.242980
  7'd6,
  7'd41,
  7'd69:b_tmp1 =20'b0111101010_1101101011; //W06_128=+0.956940 -0.290285
  7'd42:b_tmp1 =20'b0111100010_1101010100; //W07_128=+0.941544 -0.336890
  7'd7,
  7'd43:b_tmp1 =20'b0111011001_1100111100; //W08_128=+0.923880 -0.382683
  7'd44,
  7'd70:b_tmp1 =20'b0111001111_1100100101; //W09_128=+0.903989 -0.427555
  7'd8,
  7'd45:b_tmp1 =20'b0111000100_1100001111; //W10_128=+0.881921 -0.471397
  7'd46:b_tmp1 =20'b0110110111_1011111001; //W11_128=+0.857729 -0.514103
  7'd9,
  7'd47,
  7'd71:b_tmp1 =20'b0110101010_1011100100; //W12_128=+0.831470 -0.555570
  7'd48:b_tmp1 =20'b0110011011_1011001111; //W13_128=+0.803208 -0.595699
  7'd10,
```

```
7'd49:b_tmp1 =20'b0110001100_1010111011; //W14_128=+0.773010 -0.634393
7'd50,
7'd72:b_tmp1 =20'b0101111011_1010101000; //W15_128=+0.740951 -0.671559
7'd11,
7'd51:b_tmp1 =20'b0101101010_1010010110; //W16_128=+0.707107 -0.707107
7'd52:b_tmp1 =20'b0101011000_1010000101; //W17_128=+0.671559 -0.740951
7'd12,
7'd73,
7'd53:b_tmp1 =20'b0101000101_1001110100; //W18_128=+0.634393 -0.773010
7'd54:b_tmp1 =20'b0100110001_1001100101; //W19_128=+0.595699 -0.803208
7'd13,
7'd55:b_tmp1 =20'b0100011100_1001010110; //W20_128=+0.555570 -0.831470
7'd74,
7'd56:b_tmp1 =20'b0100000111_1001001001; //W21_128=+0.514103 -0.857729
7'd14,
7'd57:b_tmp1 =20'b0011110001_1000111100; //W22_128=+0.471397 -0.881921
7'd58:b_tmp1 =20'b0011011011_1000110001; //W23_128=+0.427555 -0.903989
7'd15,
7'd75,
7'd59:b_tmp1 =20'b0011000100_1000100111; //W24_128=+0.382683 -0.923880
7'd60:b_tmp1 =20'b0010101100_1000011110; //W25_128=+0.336890 -0.941544
7'd16,
7'd61:b_tmp1 =20'b0010010101_1000010110; //W26_128=+0.290285 -0.956940
7'd76,
7'd62:b_tmp1 =20'b0001111100_1000001111; //W27_128=+0.242980 -0.970031
7'd17,
7'd63:b_tmp1 =20'b0001100100_1000001010; //W28_128=+0.195090 -0.980785
7'd64:b_tmp1 =20'b0001001011_1000000110; //W29_128=+0.146730 -0.989177
7'd18,
7'd77,
7'd65:b_tmp1 =20'b0000110010_1000000010; //W30_128=+0.098017 -0.995185
7'd66:b_tmp1 =20'b0000011001_1000000001; //W31_128=+0.049068 -0.998795
7'd19:b_tmp1 =20'b0000000000_1000000000; //W32_128=+0.000000 -1.000000
7'd78:b_tmp1 =20'b1111100111_1000000001; //W33_128=-0.049068 -0.998795
7'd20:b_tmp1 =20'b1111001110_1000000010; //W34_128=-0.098017 -0.995185
7'd79,
7'd21:b_tmp1 =20'b1110011100_1000001010; //W36_128=-0.195090 -0.980785
7'd22:b_tmp1 =20'b1101101011_1000010110; //W38_128=-0.290285 -0.956940
7'd80:b_tmp1 =20'b1101010100_1000011110; //W39_128=-0.336890 -0.941544
7'd23:b_tmp1 =20'b1100111100_1000100111; //W40_128=-0.382683 -0.923880
7'd81,
7'd24:b_tmp1 =20'b1100001111_1000111100; //W42_128=-0.471397 -0.881921
7'd25:b_tmp1 =20'b1011100100_1001010110; //W44_128=-0.555570 -0.831470
7'd82:b_tmp1 =20'b1011001111_1001100101; //W45_128=-0.595699 -0.803208
7'd26:b_tmp1 =20'b1010111011_1001110100; //W46_128=-0.634393 -0.773010
7'd83,
7'd27:b_tmp1 =20'b1010010110_1010010110; //W48_128=-0.707107 -0.707107
7'd28:b_tmp1 =20'b1001110100_1010111011; //W50_128=-0.773010 -0.634393
7'd84:b_tmp1 =20'b1001100101_1011001111; //W51_128=-0.803208 -0.595699
7'd29:b_tmp1 =20'b1001010110_1011100100; //W52_128=-0.831470 -0.555570
7'd85,
7'd30:b_tmp1 =20'b1000111100_1100001111; //W54_128=-0.881921 -0.471397
7'd31:b_tmp1 =20'b1000100111_1100111100; //W56_128=-0.923880 -0.382683
7'd86:b_tmp1 =20'b1000011110_1101010100; //W57_128=-0.941544 -0.336890
7'd32:b_tmp1 =20'b1000010110_1101101011; //W58_128=-0.956940 -0.290285
7'd87,
```

```
7'd33:b_tmp1 =20'b1000001010_1110011100; //W60_128=-0.980785 -0.195090
7'd34:b_tmp1 =20'b1000000010_1111001110; //W62_128=-0.995185 -0.098017
7'd88:b_tmp1 =20'b1000000001_1111100111; //W63_128=-0.998795 -0.049068
7'd89:b_tmp1 =20'b1000000010_0000110010; //W66_128=-0.995185 +0.098017
7'd90:b_tmp1 =20'b1000001111_0001111100; //W69_128=-0.970031 +0.242980
7'd91:b_tmp1 =20'b1000100111_0011000100; //W72_128=-0.923880 +0.382683
7'd92:b_tmp1 =20'b1001001001_0100000111; //W75_128=-0.857729 +0.514103
7'd93:b_tmp1 =20'b1001110100_0101000101; //W78_128=-0.773010 +0.634393
7'd94:b_tmp1 =20'b1010101000_0101111011; //W81_128=-0.671559 +0.740951
7'd95:b_tmp1 =20'b1011100100_0110101010; //W84_128=-0.555570 +0.831470
7'd96:b_tmp1 =20'b1100100101_0111001111; //W87_128=-0.427555 +0.903989
7'd97:b_tmp1 =20'b1101101011_0111101010; //W90_128=-0.290285 +0.956940
7'd98:b_tmp1 =20'b1110110101_0111111010; //W93_128=-0.146730 +0.989177
default:b_tmp1 =20'b0111111111_0000000000; //W00_128=+1.000000 -0.000000
endcase

always @(posedge clk)
 if (enable_3)
  b_tmp2 <= b_tmp1;

assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
assign out_bi = b_tmp2[c_wordlength-1:0];

endmodule
```

Listing 11

```
// SccsId: %W% %G%
/******************************************************************************
   Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for a lookup table decoder.

 Notes   : Used to generate addresses for each coefficient, based on the
     in_Address. Addresses are dependent on one of 4 rows
     (see figures) and on the sequence length (rom_AddressSize). Each
     row gives rise to a unique address sequence based on an
     algorithm. N refers to the index of the twiddle factor, NOT the
       absolute address. Breakpoints determine where inc values change
     on line 2.

******************************************************************************/

`timescale 1ns / 100ps

module fft_coeff_dcd (clk, enable_3, in_address, out_address, nrst);

 parameter        rom_AddressSize = 1;   // Twice ROM address.
 parameter        break_point2 = 1;     // 2nd break pt line 2
 parameter        break_point3 = 1;     // 3rd break pt line 2

 input [rom_AddressSize-1:0] in_address;
 input            clk,
              nrst,
```

```
            enable_3;

output [rom_AddressSize-2:0] out_address;

wire  [rom_AddressSize-2:0] out_address;
wire  [1:0]        line_number;
wire            nrst;

reg  [rom_AddressSize-2:0] out_address_tmp;
reg  [1:0]        inc, count;
reg            rst;

// Decode which of the 4 lines are being addressed and assign it a line no.
// Only need upper two bits of in_address since 4 lines in sequence length.
assign line_number = {in_address[rom_AddressSize-1],
        in_address[rom_AddressSize-2]};

// Check for end of line and force out_address to zero on next clock edge.
always @(in_address)
 if (in_address[rom_AddressSize-3:0] == {rom_AddressSize-2{1'b1}})
  rst = 0;
 else
  rst = 1;

// Check for line number and decode appropriate out_address using algorithm
// derived by studying coefficient tables for mpys M0, M1 and M2.
always @(line_number or in_address or count)
case (line_number)
 //----------------------------------------------------------------
 2'd0: // LINE 0, inc by 2, then run the inc sequence 1,1,2,1,1,2...
  begin
     if (in_address[rom_AddressSize-3] & (|in_address[rom_AddressSize-4:0]))
      begin
       if (count == 2'd1 | count == 2'd0)
        inc = 2'd1;
       else
        inc = 2'd2;
      end
     else
      inc = 2'd2;
  end
 //----------------------------------------------------------------
 2'd1: // LINE 1, inc by 1.
  inc = 1;
 //----------------------------------------------------------------
 2'd2: // LINE 2 inc by 3, (inc by 2 at N/4+1), (inc by 1 at N/2-1).
  begin
     if (in_address[rom_AddressSize-3:0] >= break_point3)
      inc = 2'd1;          // Third stage, inc by 1.
     else if (in_address[rom_AddressSize-3:0] >= break_point2)
      inc = 2'd2;          // Second stage, inc by 2.
     else
      inc = 2'd3;          // First stage, inc by 3.
  end
 //----------------------------------------------------------------
 2'd3: // LINE 3, fixed at address 0.
```

```
 inc = 2'd0;
//-------------------------------------------------------------------
endcase

always @(posedge clk)
 if (enable_3)
 begin
  if (!nrst || !rst)    // out_address=0 at end of line or pwr Reset.
     out_address_tmp <= 0;
  else
     out_address_tmp <= out_address_tmp + inc;

  // Only count if at the correct point on line 2.
  if (in_address[rom_AddressSize-3] & (|in_address[rom_AddressSize-4:0]))
     count <= ((count == 2'd2) ? 2'd0 : count + 2'd1); // Only count to 2.
  else
     count <= 2'd0;
 end

assign out_address = out_address_tmp;
endmodule
```

Listing 12

```
// SccsId: %W% %G%
/**********************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

  Author   : Dawood Alam.

  Description: Verilog code for a configurable 2K/8K radix 2^2 + 2,
       singlepath-delay-feedback, decimation in frequency,
       (r22+2sdf DIF) Fast Fourier Transform (FFT) processor. (RTL)

  Notes   : This FFT processor computes one pair of I/Q data points every 4
       fast clk cycles. A synchronous active-low reset flushes the
         entire pipeline and resets the FFT. Therefore the next pair of
         valid inputs are assumed to be the start of the active interval
         of the next symbol. There is a latency of 2048/8192 sample
         points + 7 slow clock cycles. This equates to (2048/8192 + 7)*4
         fast clk cycles. When the out_ovf flag is raised an overflow has
         occured and saturation is performed on the intermediate
         calculation upon which the overflow has occured. If the valid_in
       flag is held low, the entire pipeline is halted and the
       valid_out flag is also held low. valid_out is also held low
       until the entire pipeline is full (after the above number of
       clock cycles).

       To Do: RAM control (MUX),
          ROM lookup (quadrant lookup),
             Change BF code for unique saturation nets for synthesis.
          ovf_detection (correct) register o/p
             ovf detection (correct) for mpy and BFs
            ROM/RAM test stuff.

 **********************************************************************************/
```

```
`timescale 1ns / 100ps

module fft_r22sdf     (in_xr,
            in_xi,
            clk,
            nrst,
              in_2k8k,
              valid_in,
            out_xr,
              out_xi,
               out_ovf,
            enable_0,
              enable_1,
              enable_2,
              enable_3,
               valid_out,
              ram_address,
              ram_enable,
              address_rom3,
              address_rom4,
            z2r_4, z2i_4,      // RAM input ports.
               z2r_5, z2i_5,      // Output data from this
               z2r_6, z2i_6,      // module.
               z2r_7, z2i_7,
               z2r_8, z2i_8,
               z2r_9, z2i_9,
               z2r_10, z2i_10,
               x1r_4, x1i_4,      // RAM output ports.
               x1r_5, x1i_5,      // Input data to this
               x1r_6, x1i_6,      // module.
               x1r_7, x1i_7,
               x1r_8, x1i_8,
               x1r_9, x1i_9,
               x1r_10, x1i_10,
              br_3,  bi_3,
              br_4,  bi_4);

// -----------------------------------------------------------------
//          Parameter definitions.
// -----------------------------------------------------------------

parameter        wordlength = 12;     // Data wordlength.
parameter        c_wordlength = 10;    // Coeff wordlength.
parameter        AddressSize = 13;     // Size of address bus.
parameter        rom_AddressSize = 13;  // ROM address bus size.
parameter        mult_scale = 3;       // Multiplier scalling:
                  // 1 = /4096, 2 = /2048,
                  // 3 = /1024, 4 = /512.
parameter        s12_wdlength = 11;    // Sectn 12 wordlength.
parameter        s11_wdlength = 12;    // Sectn 11 wordlength.
                  // s11 >= s12 >=wordlen

// -----------------------------------------------------------------
//          Input/Output ports.
// -----------------------------------------------------------------
```

```
input          clk,          // Master clock.
          nrst,          // Power-up reset.
            in_2k8k,          // 2K mode active low.
            valid_in;          // Input data valid.
input [9:0]      in_xr,          // FFT input data, I.
          in_xi;          // FFT input data, Q.

input [wordlength-1:0] x1r_4, x1i_4,      // RAM output ports.
          x1r_5, x1i_5,
            x1r_6, x1i_6,
            x1r_7, x1i_7,
            x1r_8, x1i_8,
            x1r_9, x1i_9,
            x1r_10, x1i_10;

input [c_wordlength-1:0] br_3, bi_3,
          br_4, bi_4;

output          out_ovf,          // Overflow flag.
          enable_0,          // Enable clock 0.
          enable_1,          // Enable clock 1.
          enable_2,          // Enable clock 2.
          enable_3,          // Enable clock 3.
            valid_out,          // Output data valid.
            ram_enable;

output [wordlength-1:0] out_xr,          // FFT output data, I.
          out_xi;          // FFT output data, Q.

output [wordlength-1:0] z2r_4, z2i_4,      // RAM input ports.
            z2r_5, z2i_5,
            z2r_6, z2i_6,
          z2r_7, z2i_7,
            z2r_8, z2i_8,
            z2r_9, z2i_9,
          z2r_10, z2i_10;

output [rom_AddressSize-6:0] address_rom3;
output [rom_AddressSize-4:0] address_rom4;

output [AddressSize-1:0] ram_address;

// -----------------------------------------------------------------
//          Wire/register declarations.
// -----------------------------------------------------------------

wire [1:0]        control;          // clk decode.
wire [AddressSize-1:0] address,          // FFT main address bus.
          s,          // Pipeline SRs to BFs.
            ram_address;          // RAM address bus.

wire [wordlength-1:0] x1r_0, x1i_0,      // Couples the I/Q data
          x1r_1, x1i_1,          // outputs from the
          x1r_2, x1i_2,          // memory to the
          x1r_3, x1i_3,          // respective butterfly
          x1r_4, x1i_4,          // processors, via an
```

```
x1r_5, x1i_5,      // input register.
  x1r_6, x1i_6,
  x1r_7, x1i_7,
  x1r_8, x1i_8,
  x1r_9, x1i_9,
  x1r_10, x1i_10,

x2r_0, x2i_0,      // Couples the I/Q data
  x2r_1, x2i_1,      // outputs from BF2I
  x2r_2, x2i_2,      // to the I/Q inputs of
  x2r_3, x2i_3,      // BF2II. Also connects
x2r_4, x2i_4,      // the I/Q ouputs of the
x2r_5, x2i_5,      // complex multiplier
x2r_6, x2i_6,      // to the inputs of the
x2r_7, x2i_7,      // next radix 2^2 stage.
  x2r_8, x2i_8,
  x2r_9, x2i_9,
  x2r_10, x2i_10;

reg [wordlength-1:0]  x1r_4_tmp, x1i_4_tmp,  // Registered inputs
        x1r_5_tmp, x1i_5_tmp,  // from RAM.
          x1r_6_tmp, x1i_6_tmp,
          x1r_7_tmp, x1i_7_tmp,
          x1r_8_tmp, x1i_8_tmp,
          x1r_9_tmp, x1i_9_tmp,
          x1r_10_tmp, x1i_10_tmp;

wire  [s11_wdlength-1:0] x1r_11, x1i_11,      // Different bit-widths
        x2r_11, x2i_11;      // for I/Q lines, but
wire  [s12_wdlength-1:0] x1r_12, x1i_12;      // similar to the above.

wire  [wordlength-1:0]  ar_0, ai_0,        // Couples the I/Q data
        ar_1, ai_1,        // outputs of the
          ar_2, ai_2,        // previous radix 2^2
          ar_3, ai_3,        // stage into the
        ar_4, ai_4,        // complex multiplier
          ar_5, ai_5;        // of the next stage.

wire  [c_wordlength-1:0] br_0, bi_0,        // Couples the I/Q
        br_1, bi_1,        // coefficient outputs
          br_2, bi_2,        // from the ROM demapper
          br_3, bi_3,        // to the complex
        br_4, bi_4,        // multiplier.
          br_5, bi_5;

wire  [wordlength-1:0]  z2r_0, z2i_0,
        z2r_1, z2i_1,
          z2r_2, z2i_2,
        z2r_3, z2i_3;

reg  [wordlength-1:0]  z2r_4, z2i_4,      // Registered outputs
        z2r_5, z2i_5,      // to RAM.
          z2r_6, z2i_6,
        z2r_7, z2i_7,
          z2r_8, z2i_8,
          z2r_9, z2i_9;
```

```
wire [wordlength-1:0] z2r_10, z2i_10; // WILL CHANGE WHEN RAM RIGHT 2 rg

wire [wordlength-1:0] z2r_4_tmp, z2i_4_tmp, // Couple the I/Q data
          z2r_5_tmp, z2i_5_tmp, // outputs of each BF
            z2r_6_tmp, z2i_6_tmp, // processor to their
          z2r_7_tmp, z2i_7_tmp, // respective memory
            z2r_8_tmp, z2i_8_tmp, // inputs via an output
            z2r_9_tmp, z2i_9_tmp, // register.
          z2r_10_tmp, z2i_10_tmp;

wire [s11_wdlength-1:0] z2r_11, z2i_11;    // Different bit-widths
wire [s12_wdlength-1:0] z2r_12, z2i_12;    // for the 1st 2 stages.

wire [rom_AddressSize-8:0] address_rom2;   // Couples the address
wire [rom_AddressSize-6:0] address_rom3;   // decoders outputs to
wire [rom_AddressSize-4:0] address_rom4;   // respective ROMs.
wire [rom_AddressSize-2:0] address_rom5;

wire [rom_AddressSize-7:0] dcd_address2;   // Couples part of the
wire [rom_AddressSize-5:0] dcd_address3;   // address bus to the
wire [rom_AddressSize-3:0] dcd_address4;   // coefficient decoder.
wire [rom_AddressSize-1:0] dcd_address5;

wire        ovf_0, ovf_1,    // Couples overflow
          ovf_2, ovf_3,    // flag outputs from
            ovf_4, ovf_5,    // each butterfly
            ovf_6, ovf_7,    // processor and complex
            ovf_8, ovf_9,    // multiplier into one
            ovf_10, ovf_11,    // overflow status flag
            ovf_12, ovf_13,    // called "out_ovf".
            ovf_14, ovf_15,
            ovf_16, ovf_17,
            ovf_18;

wire        clk,
          nrst,
            in_2k8k,
          ovf_2k,
            out_ovf,
            enable_0,
            enable_1,
            enable_2,
            enable_3,
          ram_enable;     // RAM enable signal.

reg         ovf_tmp1,
          ovf_tmp2,
          fft_cycle_complete,   // End of 1st FFT cycle.
            output_valid;     // Output valid flag.
reg [3:0]      pipeline_count;     // Counts pipeline regs.
reg [AddressSize-1:0] q, t;
reg [1:0]       r;
reg [wordlength-1:0] x1r_0_reg, x1i_0_reg,
          xr_tmp2,      // Output data reg, I.
            xi_tmp2;      // Output data reg, Q.
reg [s12_wdlength-1:0] in_xr_tmp, in_xi_tmp;
```

```
reg [9:0]      xr_reg,        // Input data reg, I.
        xi_reg;        // Input data reg, Q.
reg [wordlength-1:0] x2r_10_tmp2, x2i_10_tmp2,
        x2r_10_tmp3, x2i_10_tmp3;

wire [wordlength-1:0] xr_tmp1,        // Final BF2I(0) out, I.
        xi_tmp1;        // Final BF2I(0) out, Q.
wire [wordlength-1:0] x2r_10_tmp1, x2i_10_tmp1;
wire [s12_wdlength-1:0] x2r_11_tmp, x2i_11_tmp;

// ---------------------------------------------------------------
//  Address decoders/Quadrant mappers + pipeline shift registers.
// ---------------------------------------------------------------

/* fft_sr_addr #(rom_AddressSize-6, 3) sr_addr_2
              (clk, enable_3,
              address[6:0],  // Input.
              dcd_address2);  // Output.

fft_coeff_dcd #(rom_AddressSize-6, 11, 21)
coeff_dcd_2 (clk, enable_3, dcd_address2, address_rom2, nrst); */

// ---------------------------------------------------------------

fft_sr_addr #(rom_AddressSize-4, 2) sr_addr_3
             (clk, enable_3,
             address[8:0],   // Input.
             dcd_address3);  // Output.

fft_coeff_dcd #(rom_AddressSize-4, 43, 85)
coeff_dcd_3 (clk, enable_3, dcd_address3, address_rom3, nrst);

// ---------------------------------------------------------------

fft_sr_addr #(rom_AddressSize-2, 1) sr_addr_4
             (clk, enable_3,
             address[10:0],   // Input.
             dcd_address4);   // Output.

fft_coeff_dcd #(rom_AddressSize-2, 171, 341)
coeff_dcd_4 (clk, enable_3, dcd_address4, address_rom4, nrst);

// ---------------------------------------------------------------

/* fft_coeff_dcd #(rom_AddressSize, 683, 1365)
coeff_dcd_5 (clk, enable_3, address, address_rom5, nrst); */

// ---------------------------------------------------------------
//          ROM lookup tables.
// ---------------------------------------------------------------

fft_hardwired_lu0 #(c_wordlength, rom_AddressSize-10)// Case table instance
rom0 (clk, enable_3, address[2:0], br_0, bi_0);   // for a hardwired ROM.

fft_hardwired_lu1 #(c_wordlength, rom_AddressSize-8) // Case table instance
rom1 (clk, enable_3, address[4:0], br_1, bi_1);   // for a hardwired ROM.
```

```
fft_hardwired_lu2 #(c_wordlength, rom_AddressSize-6) // Case table instance
 rom2 (clk, enable_3, address[6:0], br_2, bi_2);   // for a hardwired ROM.

/*fft_hardwired_lu3 #(c_wordlength, rom_AddressSize-4) // Case table instance
 rom3 (clk, enable_3, address[8:0], br_3, bi_3);   // for a hardwired ROM.*/

/*fft_hardwired_lu3 #(c_wordlength, rom_AddressSize-5)// Case table instance
 rom3 (clk, enable_3, address_rom3, br_3, bi_3);   // for a hardwired ROM.*/

/*fft_rom #(c_wordlength, rom_AddressSize-6,
   "../../../fft/src/lookup_tables/lu_10bit_128pt_scale1")
 rom2 (address[6:0], br_2, bi_2); // 128 addresses x 20 bits, no decode. */

/*fft_rom #(c_wordlength, rom_AddressSize-7,
   "../../../fft/src/lookup_tables/lu_10bit_128pt_scale1")
 rom2 (address_rom2, br_2, bi_2); // 64 addresses x 20 bits, coeff decode. */

/*fft_rom #(c_wordlength, rom_AddressSize-4,
   "../../../fft/src/lookup_tables/lu_10bit_512pt_scale1")
 rom3 (address[8:0], br_3, bi_3); // 512 addresses x 20 bits, no decode. */

/* fft_rom #(c_wordlength, rom_AddressSize-5,
   "../../../fft/src/lookup_tables/lu_10bit_512pt_scale1")
 rom3 (clk, enable_3, address_rom3, br_3, bi_3); // 256 addresses x 20 bits.*/

/*fft_rom #(c_wordlength, rom_AddressSize-2,
   "../../../fft/src/lookup_tables/lu_10bit_2048pt_scale1")
 rom4 (address[10:0], br_4, bi_4); // 2048 addresses x 20 bits, no decode. */

/* fft_rom #(c_wordlength, rom_AddressSize-3,
   "../../../fft/src/lookup_tables/lu_10bit_2048pt_scale1")
 rom4 (clk, enable_3, address_rom4, br_4, bi_4); // 1024 addresses x 20 bits.*/

/*fft_rom #(c_wordlength, rom_AddressSize,
   "../../../fft/src/lookup_tables/lu_10bit_8192pt_scale1")
 rom5 (address, br_5, bi_5); // 8192 addresses x 20 bits, no decode. */

/* fft_rom #(c_wordlength, rom_AddressSize-1,
   "../../../fft/src/lookup_tables/lu_10bit_8192pt_scale1")
 rom5 (clk, enable_3, address_rom5, br_5, bi_5); // 4096 addresses x 20 bits.*/

 // -------------------------------------------------------------------
 //    Section 12 and 11, tail end of FFT pipeline (input stage).
 //  Section 12 is 11 bits wide and incorporates the 2K/8K control logic.
 // -------------------------------------------------------------------

 always @(xr_reg or xi_reg or in_2k8k or x2r_10_tmp1 or x2i_10_tmp1
                 or x2r_10_tmp3 or x2i_10_tmp3)
  if (!in_2k8k) // Configuring for 2K mode.
   begin
     x2r_10_tmp2 = x2r_10_tmp3;
     x2i_10_tmp2 = x2i_10_tmp3;
     in_xr_tmp = 0;
     in_xi_tmp = 0;
   end
  else    // Configuring for 8K mode.
```

```
 begin
   x2r_10_tmp2 = x2r_10_tmp1;
   x2i_10_tmp2 = x2i_10_tmp1;
   // Sign extend from 10 bits, as section 12 is s12_wdlength bits.
   in_xr_tmp = {{(s12_wdlength-9){xr_reg[9]}},xr_reg[8:0]};
   in_xi_tmp = {{(s12_wdlength-9){xi_reg[9]}},xi_reg[8:0]};
 end

always @(posedge clk) // Pipeline register to enable correct operation in
 if (enable_3)    // 2K mode without retiming the entire pipeline since
  begin      // 8K mode introduces 1 additional pipeline register.
    // Sign extend 10 bit inputs to wordlength bit inputs.
    // for bypass lines into stage 5.
    x2r_10_tmp3 <= {{(wordlength-9){xr_reg[9]}},xr_reg[8:0]};
    x2i_10_tmp3 <= {{(wordlength-9){xi_reg[9]}},xi_reg[8:0]};
  end

assign x2r_10 = x2r_10_tmp2;
assign x2i_10 = x2i_10_tmp2;

// Sign extend from s12_wdlength bits to s11_wdlength bits between
// sections 12 and 11. Uncomment below if s_11 < > s_12.
assign x2r_11 = {{(s11_wdlength-s12_wdlength+1)
      {x2r_11_tmp[s12_wdlength-1]}},x2r_11_tmp[s12_wdlength-2:0]};
assign x2i_11 = {{(s11_wdlength-s12_wdlength+1)
      {x2i_11_tmp[s12_wdlength-1]}},x2i_11_tmp[s12_wdlength-2:0]};
// Uncomment below if s_11 = s_12.
/* assign x2r_11 = x2r_11_tmp;
 assign x2i_11 = x2i_11_tmp; */

fft_bf2I #(s12_wdlength) bf2I_6
        (clk, enable_1,
          x1r_12, x1i_12, in_xr_tmp, in_xi_tmp,  // Ext In.
        s[12],
        x2r_11_tmp, x2i_11_tmp, z2r_12, z2i_12, // Outputs.
        ovf_18);

/* fft_ram #(s12_wdlength, 12) ram_12 (clk, enable_1, enable_3,
            ram_address[11:0],     // 4096 addrs.
            z2r_12, z2i_12,     // Inputs.
            x1r_12, x1i_12);     // Outputs. */

 fft_bf2II #(s11_wdlength) bf2II_6
        (clk, enable_1,
          x1r_11, x1i_11, x2r_11, x2i_11,     // Inputs.
        s[11], s[12],
        ar_5, ai_5, z2r_11, z2i_11,      // Outputs.
        ovf_17);

fft_sr_1bit #(1) sr_1bit_11 (clk, enable_3, address[11], s[11]); // SR 11.
fft_sr_1bit #(1) sr_1bit_12 (clk, enable_3, address[12], s[12]); // SR 12.

/* fft_ram #(s11_wdlength, 11) ram_11 (clk, enable_1, enable_3,
            ram_address[10:0],     // 2048 addrs.
            z2r_11, z2i_11,     // Inputs.
            x1r_11, x1i_11);     // Outputs. */
```

```
// ------------------------------------------------------------------
//          Section 10 and 9.
// ------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m5
        (clk, control,
            ar_5, ai_5, br_5, bi_5,       // Inputs.
         x2r_10_tmp1, x2i_10_tmp1,      // Outputs.
         ovf_16);

fft_bf2I #(wordlength) bf2I_5 (clk, enable_1,
            x1r_10, x1i_10,    // Inputs.
              x2r_10, x2i_10,
            s[10],
            x2r_9, x2i_9,         // Outputs.
              z2r_10, z2i_10,
            ovf_15);

fft_bf2II #(wordlength) bf2II_5 (clk, enable_1,
            x1r_9_tmp, x1i_9_tmp,      // Inputs.
              x2r_9, x2i_9,
            s[9], s[10],
            ar_4, ai_4,          // Outputs.
              z2r_9_tmp, z2i_9_tmp,
            ovf_14);

fft_sr_1bit #(2) sr_1bit_9 (clk, enable_3, address[9], s[9]); // SR 9.
fft_sr_1bit #(2) sr_1bit_10 (clk, enable_3, address[10], s[10]); // SR 10.


// ------------------------------------------------------------------
//          Section 8 and 7.
// ------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m4
        (clk, control,
            ar_4, ai_4, br_4, bi_4,       // Inputs.
         x2r_8, x2i_8,           // Outputs.
         ovf_13);

fft_bf2I #(wordlength) bf2I_4 (clk, enable_1,
            x1r_8_tmp, x1i_8_tmp,      // Inputs.
              x2r_8, x2i_8,
            s[8],
            x2r_7, x2i_7,         // Outputs.
              z2r_8_tmp, z2i_8_tmp,
            ovf_12);

fft_bf2II #(wordlength) bf2II_4 (clk, enable_1,
            x1r_7_tmp, x1i_7_tmp,      // Inputs.
              x2r_7, x2i_7,
            s[7], s[8],
            ar_3, ai_3,          // Outputs.
              z2r_7_tmp, z2i_7_tmp,
            ovf_11);
```

```
fft_sr_1bit #(3) sr_1bit_7 (clk, enable_3, address[7], s[7]); // SR 7.
fft_sr_1bit #(3) sr_1bit_8 (clk, enable_3, address[8], s[8]); // SR 8.

// -------------------------------------------------------------------
//          Section 6 and 5.
// -------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m3
        (clk, control,
         ar_3, ai_3, br_3, bi_3,      // Inputs.
        x2r_6, x2i_6,           // Outputs.
        ovf_10);

fft_bf2I #(wordlength) bf2I_3 (clk, enable_1,
           x1r_6_tmp, x1i_6_tmp,      // Inputs.
             x2r_6, x2i_6,
           s[6],
           x2r_5, x2i_5,         // Outputs.
             z2r_6_tmp, z2i_6_tmp,
           ovf_9);

fft_bf2II #(wordlength) bf2II_3 (clk, enable_1,
           x1r_5_tmp, x1i_5_tmp,      // Inputs.
             x2r_5, x2i_5,
           s[5], s[6],
           ar_2, ai_2,          // Outputs.
             z2r_5_tmp, z2i_5_tmp,
           ovf_8);

fft_sr_1bit #(4) sr_1bit_5 (clk, enable_3, address[5], s[5]); // SR 5.
fft_sr_1bit #(4) sr_1bit_6 (clk, enable_3, address[6], s[6]); // SR 6.

// -------------------------------------------------------------------
//          Section 4 and 3.
// -------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m2
        (clk, control,
            ar_2, ai_2, br_2, bi_2,      // Inputs.
        x2r_4, x2i_4,           // Outputs.
        ovf_7);

fft_bf2I #(wordlength) bf2I_2 (clk, enable_1,
           x1r_4_tmp, x1i_4_tmp,      // Inputs.
             x2r_4, x2i_4,
           s[4],
           x2r_3, x2i_3,         // Outputs.
             z2r_4_tmp, z2i_4_tmp,
           ovf_6);

fft_bf2II #(wordlength) bf2II_2 (clk, enable_1,
           x1r_3, x1i_3,        // Inputs.
             x2r_3, x2i_3,
           s[3], s[4],
           ar_1, ai_1,          // Outputs.
```

```
          z2r_3, z2i_3,
        ovf_5);

fft_sr_1bit #(5) sr_1bit_3 (clk, enable_3, address[3], s[3]); // SR 3.
fft_sr_1bit #(5) sr_1bit_4 (clk, enable_3, address[4], s[4]); // SR 4.

fft_sr_iq #(wordlength, 8) sr_iq_3 (clk, enable_3,    // Length = 8.
          z2r_3, z2i_3,     // Inputs.
          x1r_3, x1i_3);    // Outputs.

// -------------------------------------------------------------------
//          Section 2 and 1.
// -------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m1
       (clk, control,
          ar_1, ai_1, br_1, bi_1,      // Inputs.
        x2r_2, x2i_2,           // Outputs.
        ovf_4);

fft_bf2I #(wordlength) bf2I_1 (clk, enable_1,
         x1r_2, x1i_2,      // Inputs.
           x2r_2, x2i_2,
         s[2],
         x2r_1, x2i_1,      // Outputs.
           z2r_2, z2i_2,
         ovf_3);

fft_sr_iq #(wordlength, 4) sr_iq_2 (clk, enable_3,    // Length = 4.
          z2r_2, z2i_2,     // Inputs.
          x1r_2, x1i_2);    // Outputs.

fft_bf2II #(wordlength) bf2II_1 (clk, enable_1,
         x1r_1, x1i_1,      // Inputs.
           x2r_1, x2i_1,
         s[1], s[2],
         ar_0, ai_0,       // Outputs.
           z2r_1, z2i_1,
         ovf_2);

assign s[1] = ~address[1]; // Invert s[1] (see count sequence), SR1 not req.
//fft_sr_1bit #(6) sr_1bit_1 (clk, enable_3, address[1], s[1]); // SR 1.
fft_sr_1bit #(6) sr_1bit_2 (clk, enable_3, address[2], s[2]); // SR 2.

fft_sr_iq #(wordlength, 2) sr_iq_1 (clk, enable_3,    // Length = 2.
          z2r_1, z2i_1,     // Inputs.
          x1r_1, x1i_1);    // Outputs.

// -------------------------------------------------------------------
//  Section 0, front end of FFT pipeline (output stage), mult_scale=4.
// -------------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, 4) m0
       (clk, control,
          ar_0, ai_0, br_0, bi_0,      // Inputs.
        x2r_0, x2i_0,           // Outputs.
```

```
        ovf_1);

    fft_bf2I #(wordlength) bf2I_0 (clk, enable_1,
            x1r_0, x1i_0,       // Inputs.
              x2r_0, x2i_0,
            s[0],
            xr_tmp1, xi_tmp1,       // Outputs.
              z2r_0, z2i_0,
            ovf_0);

     assign s[0] = ~address[0]; // Invert s[0] (see count sequence), SR0 not req.
    //fft_sr_1bit #(7) sr_1bit_0 (clk, enable_3, address[0], s[0]); // SR 0.

     // Last stage should be just a single register as only 1 location needed.
     always @(posedge clk) // No reset required as data clocked through registers.
      if (enable_3)
       begin
         x1r_0_reg <= z2r_0;
         x1i_0_reg <= z2i_0;
       end

     assign x1r_0 = x1r_0_reg;
     assign x1i_0 = x1i_0_reg;

     // ----------------------------------------------------------------
     //        Register Inputs/Outputs.
     // ----------------------------------------------------------------

     `ifdef BIN_SHIFT
      always @(posedge clk)      // Registered inputs.
       if (enable_3 && !address[0])   // == freq bin shift by pi.
        begin
          xr_reg <= in_xr;
          xi_reg <= in_xi;
        end
       else if (enable_3 && address[0]) // == freq bin shift by pi.
        begin
          xr_reg <= ~in_xr + 1'b1;   // This is equivalent to multiplying by
          xi_reg <= ~in_xi + 1'b1;   // exp(-j * pi * n) == (-1)^n.
        end
      `else
       always @(posedge clk)      // Registered inputs.
        if (enable_3)
         begin
           xr_reg <= in_xr;
           xi_reg <= in_xi;
         end
      `endif

       always @(posedge clk)      // Registered outputs.
        if (enable_3)
         begin
           xr_tmp2 <= xr_tmp1;
           xi_tmp2 <= xi_tmp1;
         end
```

```
assign out_xr = xr_tmp2;
assign out_xi = xi_tmp2;

always @(posedge clk)      // RAMs are latched on outputs so no
 begin              // need to enable.
  z2r_4 <= z2r_4_tmp;     // Register FFT outputs to RAM.
  z2i_4 <= z2i_4_tmp;
  z2r_5 <= z2r_5_tmp;
  z2i_5 <= z2i_5_tmp;
  z2r_6 <= z2r_6_tmp;
  z2i_6 <= z2i_6_tmp;
  z2r_7 <= z2r_7_tmp;
  z2i_7 <= z2i_7_tmp;
  z2r_8 <= z2r_8_tmp;
  z2i_8 <= z2i_8_tmp;
  z2r_9 <= z2r_9_tmp;
  z2i_9 <= z2i_9_tmp;
//  z2r_10 <= z2r_10_tmp;
//  z2i_10 <= z2i_10_tmp;
  x1r_4_tmp <= x1r_4;     // Register FFT inputs from RAM.
  x1i_4_tmp <= x1i_4;
  x1r_5_tmp <= x1r_5;
  x1i_5_tmp <= x1i_5;
  x1r_6_tmp <= x1r_6;
  x1i_6_tmp <= x1i_6;
  x1r_7_tmp <= x1r_7;
  x1i_7_tmp <= x1i_7;
  x1r_8_tmp <= x1r_8;
  x1i_8_tmp <= x1i_8;
  x1r_9_tmp <= x1r_9;
  x1i_9_tmp <= x1i_9;
//  x1r_10_tmp <= x1r_10;
//  x1i_10_tmp <= x1i_10;
 end


// ---------------------------------------------------------------------
//         Synchronous butterfly controller.
// ---------------------------------------------------------------------

always @(posedge clk)
 if (!nrst)             // Synchronous power-up reset.
  q <= 0;
 else if (enable_3)
  q <= q + 1'b1;

assign address = q;

// ---------------------------------------------------------------------
//         Synchronous RAM address generator.
// ---------------------------------------------------------------------

always @(posedge clk)
 if (!nrst)             // Synchronous power-up reset.
  t <= 0;
 else if (enable_2)
```

```
 t <= t + 1'b1;

assign ram_address = t;

assign ram_enable = enable_3 || enable_2;     // ram enable signal.
// ----------------------------------------------------------------------
//          valid_out status flag generation.
// ----------------------------------------------------------------------

always @(posedge clk)
 if (!nrst)
  fft_cycle_complete <= 1'b0; // Detect end of 1st fft cycle i.e. 2K or 8K.
 else if ((~in_2k8k && &address[10:0]) || (in_2k8k && &address[12:0]))
  fft_cycle_complete <= 1'b1;
 else
  fft_cycle_complete <= fft_cycle_complete;

always @(posedge clk)    // Account for pipeline and I/O registers.
 if (!nrst)
  pipeline_count <= 4'b0;   // Stop at pipeline_depth - 1.
 else if (enable_3 && fft_cycle_complete & pipeline_count < 8)//pipe depth=8
  pipeline_count <= pipeline_count + 1'b1;

always @(posedge clk)      // Test if the pipeline is full and the input
 if (!nrst)         // is valid before asserting valid_out.
  output_valid <= 1'b0;
 else if (enable_2 && pipeline_count[3])
  output_valid <= 1'b1;
 else
  output_valid <= 1'b0;

assign valid_out = output_valid;

// ----------------------------------------------------------------------
//     Fast 40 MHz clock decoder and valid_in control.
// ----------------------------------------------------------------------

always @(posedge clk)
 if (!nrst)              // Synchronous power-up reset.
  r <= 0;
 else if (valid_in)           // Count if input data valid.
  r <= r + 1'b1;

assign control = {valid_in & r[1],valid_in & r[0]};

assign enable_0 = valid_in & (~r[1] & ~r[0]);  // Gate valid_in with
assign enable_1 = valid_in & (~r[1] & r[0]);  // decoded enable signals
assign enable_2 = valid_in & ( r[1] & ~r[0]);  // to control all reg's.
assign enable_3 = valid_in & ( r[1] & r[0]);

// ----------------------------------------------------------------------
// Overflow detection, OR overflows from each stage to give overflow flag.
// ----------------------------------------------------------------------

assign ovf_2k = ovf_0 || ovf_1 || ovf_2 || ovf_3 || ovf_4 ||
      ovf_5 || ovf_6 || ovf_7 || ovf_8 || ovf_9 ||
```

```
      ovf_10 || ovf_11 || ovf_12 || ovf_13 || ovf_14 ||
    ovf_15;

// 2k/8k Overflow flag configuration.
always @(in_2k8k or ovf_16 or ovf_17 or ovf_18 or ovf_2k)
 if (in_2k8k)
  ovf_tmp1 = ovf_2k || ovf_16 || ovf_17 || ovf_18;
 else
  ovf_tmp1 = ovf_2k;

always @(posedge clk)             // Register overflow
 if (enable_3 && fft_cycle_complete)       // flag to change when
  ovf_tmp2 <= ovf_tmp1;           // I/Q samples are valid
                     // from FFT processor.
assign out_ovf = ovf_tmp2;

`ifdef OVERFLOW_DEBUG
// Debug code to display overflow output of a particular instance.
// Concurrently monitor overflow flag and halt on overflow.
always @(out_ovf) // ovf_x wires are all registered at lower level.
 if (out_ovf)
  begin
    $display ("Overflow has occurred, type . to continue.");
    $display ("Overflow flag, out_ovf = ",out_ovf);
    if (ovf_18) $display ("Overflow on port ovf_18");
    if (ovf_17) $display ("Overflow on port ovf_17");
    if (ovf_16) $display ("Overflow on port ovf_16");
    if (ovf_15) $display ("Overflow on port ovf_15");
    if (ovf_14) $display ("Overflow on port ovf_14");
    if (ovf_13) $display ("Overflow on port ovf_13");
    if (ovf_12) $display ("Overflow on port ovf_12");
    if (ovf_11) $display ("Overflow on port ovf_11");
    if (ovf_10) $display ("Overflow on port ovf_10");
    if (ovf_9) $display ("Overflow on port ovf_9");
    if (ovf_8) $display ("Overflow on port ovf_8");
    if (ovf_7) $display ("Overflow on port ovf_7");
    if (ovf_6) $display ("Overflow on port ovf_6");
    if (ovf_5) $display ("Overflow on port ovf_5");
    if (ovf_4) $display ("Overflow on port ovf_4");
    if (ovf_3) $display ("Overflow on port ovf_3");
    if (ovf_2) $display ("Overflow on port ovf_2");
    if (ovf_1) $display ("Overflow on port ovf_1");
    if (ovf_0) $display ("Overflow on port ovf_0");
    $stop;
  end
 `endif
endmodule
```

Listing 13

```
// SccsId: %W% %G%
/*******************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for the window lookup table, used to determine the
```

variance of the data and hence the F_ratio.

Notes :

```
*******************************************************************************/

`timescale 1ns / 100ps

module fft_window_lu (clk, enable_3, in_address, out_data);

parameter        r_wordlength = 10;    // Data wordlength.
parameter        lu_AddressSize = 13;   // Address bus size.

input        clk,
         enable_3;
input [lu_AddressSize-1:0] in_address;

output [r_wordlength-1:0] out_data;

reg  [r_wordlength-1:0] data_tmp1,
         data_tmp2;

always @(in_address)
casez (in_address)
13'b0000000000000 : data_tmp1 = 10'b1000000000;

13'b0000000000001 : data_tmp1 = 10'b0000000000;

13'b0000000000010 : data_tmp1 = 10'b0000100111;

13'b0000000000011 : data_tmp1 = 10'b0000111110;

13'b0000000000100 : data_tmp1 = 10'b0001001110;

13'b0000000000101 : data_tmp1 = 10'b0001011011;

13'b0000000000110 : data_tmp1 = 10'b0001100110;

13'b0000000000111 : data_tmp1 = 10'b0001101110;

13'b0000000001000 : data_tmp1 = 10'b0001110110;

13'b0000000001001 : data_tmp1 = 10'b0001111101;

13'b0000000001010 : data_tmp1 = 10'b0010000011;

13'b0000000001011 : data_tmp1 = 10'b0010001000;

13'b0000000001100 : data_tmp1 = 10'b0010001101;

13'b0000000001101 : data_tmp1 = 10'b0010010001;

13'b0000000001110 : data_tmp1 = 10'b0010010110;

13'b0000000001111 : data_tmp1 = 10'b0010011010;
```

```
13'b0000000010000 : data_tmp1 = 10'b0010011101;
13'b0000000010001 : data_tmp1 = 10'b0010100001;
13'b0000000010010 : data_tmp1 = 10'b0010100100;
13'b0000000010011 : data_tmp1 = 10'b0010100111;
13'b0000000010100 : data_tmp1 = 10'b0010101010;
13'b0000000010101 : data_tmp1 = 10'b0010101101;
13'b0000000010110 : data_tmp1 = 10'b0010101111;
13'b0000000010111 : data_tmp1 = 10'b0010110010;
13'b0000000011000 : data_tmp1 = 10'b0010110100;
13'b0000000011001 : data_tmp1 = 10'b0010110111;
13'b0000000011010 : data_tmp1 = 10'b0010111001;
13'b0000000011011 : data_tmp1 = 10'b0010111011;
13'b0000000011100 : data_tmp1 = 10'b0010111101;
13'b0000000011101 : data_tmp1 = 10'b0010111111;
13'b0000000011110 : data_tmp1 = 10'b0011000001;
13'b0000000011111 : data_tmp1 = 10'b0011000011;
13'b0000000100000 : data_tmp1 = 10'b0011000101;
13'b0000000100001 : data_tmp1 = 10'b0011000110;
13'b0000000100010 : data_tmp1 = 10'b0011001000;
13'b0000000100011 : data_tmp1 = 10'b0011001010;
13'b0000000100100 : data_tmp1 = 10'b0011001011;
13'b0000000100101 : data_tmp1 = 10'b0011001101;
13'b0000000100110 : data_tmp1 = 10'b0011001110;
13'b0000000100111 : data_tmp1 = 10'b0011010000;
13'b0000000101000 : data_tmp1 = 10'b0011010001;
13'b0000000101001 : data_tmp1 = 10'b0011010011;
13'b0000000101010 : data_tmp1 = 10'b0011010100;
13'b0000000101011 : data_tmp1 = 10'b0011010101;
```

```
13'b0000000101100 : data_tmp1 = 10'b0011010111;
13'b0000000101101 : data_tmp1 = 10'b0011011000;
13'b0000000101110 : data_tmp1 = 10'b0011011001;
13'b0000000101111 : data_tmp1 = 10'b0011011010;
13'b0000000110000 : data_tmp1 = 10'b0011011100;
13'b0000000110001 : data_tmp1 = 10'b0011011101;
13'b0000000110010 : data_tmp1 = 10'b0011011110;
13'b0000000110011 : data_tmp1 = 10'b0011011111;
13'b0000000110100 : data_tmp1 = 10'b0011100000;
13'b0000000110101 : data_tmp1 = 10'b0011100001;
13'b0000000110110 : data_tmp1 = 10'b0011100010;
13'b0000000110111 : data_tmp1 = 10'b0011100011;
13'b0000000111000 : data_tmp1 = 10'b0011100100;
13'b0000000111001 : data_tmp1 = 10'b0011100101;
13'b0000000111010 : data_tmp1 = 10'b0011100110;
13'b0000000111011 : data_tmp1 = 10'b0011100111;
13'b0000000111100 : data_tmp1 = 10'b0011101000;
13'b0000000111101 : data_tmp1 = 10'b0011101001;
13'b0000000111110 : data_tmp1 = 10'b0011101010;
13'b0000000111111 : data_tmp1 = 10'b0011101011;
13'b0000001000000 : data_tmp1 = 10'b0011101100;
13'b0000001000001 : data_tmp1 = 10'b0011101101;
13'b0000001000010 : data_tmp1 = 10'b0011101110;
13'b0000001000011 : data_tmp1 = 10'b0011101111;
13'b0000001000100 : data_tmp1 = 10'b0011101111;
13'b0000001000101 : data_tmp1 = 10'b0011110000;
13'b0000001000110 : data_tmp1 = 10'b0011110001;
13'b0000001000111 : data_tmp1 = 10'b0011110010;
13'b000000100100z : data_tmp1 = 10'b0011110011;
```

```
13'b0000001001010 : data_tmp1 = 10'b0011110100;
13'b0000001001011 : data_tmp1 = 10'b0011110101;
13'b000000100110z : data_tmp1 = 10'b0011110110;
13'b0000001001110 : data_tmp1 = 10'b0011110111;
13'b0000001001111 : data_tmp1 = 10'b0011111000;
13'b000000101000z : data_tmp1 = 10'b0011111001;
13'b0000001010010 : data_tmp1 = 10'b0011111010;
13'b0000001010011 : data_tmp1 = 10'b0011111011;
13'b0000001010100 : data_tmp1 = 10'b0011111011;
13'b0000001010101 : data_tmp1 = 10'b0011111100;
13'b000000101011z : data_tmp1 = 10'b0011111101;
13'b0000001011000 : data_tmp1 = 10'b0011111110;
13'b0000001011001 : data_tmp1 = 10'b0011111111;
13'b0000001011010 : data_tmp1 = 10'b0011111111;
13'b0000001011011 : data_tmp1 = 10'b0100000000;
13'b000000101110z : data_tmp1 = 10'b0100000001;
13'b000000101111z : data_tmp1 = 10'b0100000010;
13'b0000001100000 : data_tmp1 = 10'b0100000011;
13'b0000001100001 : data_tmp1 = 10'b0100000100;
13'b0000001100010 : data_tmp1 = 10'b0100000100;
13'b0000001100011 : data_tmp1 = 10'b0100000101;
13'b0000001100100 : data_tmp1 = 10'b0100000101;
13'b0000001100101 : data_tmp1 = 10'b0100000110;
13'b0000001100110 : data_tmp1 = 10'b0100000110;
13'b0000001100111 : data_tmp1 = 10'b0100000111;
13'b000000110100z : data_tmp1 = 10'b0100001000;
13'b000000110101z : data_tmp1 = 10'b0100001001;
13'b000000110110z : data_tmp1 = 10'b0100001010;
13'b000000110111z : data_tmp1 = 10'b0100001011;
13'b000000111000z : data_tmp1 = 10'b0100001100;
13'b000000111001z : data_tmp1 = 10'b0100001101;
```

```
13'b000000111010z : data_tmp1 = 10'b0100001110;
13'b000000111011z : data_tmp1 = 10'b0100001111;
13'b000000111100z : data_tmp1 = 10'b0100010000;
13'b000000111101z : data_tmp1 = 10'b0100010001;
13'b000000111110z : data_tmp1 = 10'b0100010010;
13'b0000001111110 : data_tmp1 = 10'b0100010010;
13'b0000001111111 : data_tmp1 = 10'b0100010011;
13'b0000010000000 : data_tmp1 = 10'b0100010011;
13'b0000010000001 : data_tmp1 = 10'b0100010100;
13'b0000010000010 : data_tmp1 = 10'b0100010100;
13'b0000010000011 : data_tmp1 = 10'b0100010101;
13'b0000010000100 : data_tmp1 = 10'b0100010101;
13'b00000100001z1 : data_tmp1 = 10'b0100010110;
13'b0000010000110 : data_tmp1 = 10'b0100010110;
13'b000001000100z : data_tmp1 = 10'b0100010111;
13'b000001000101z : data_tmp1 = 10'b0100011000;
13'b0000010001100 : data_tmp1 = 10'b0100011000;
13'b0000010001101 : data_tmp1 = 10'b0100011001;
13'b0000010001110 : data_tmp1 = 10'b0100011001;
13'b0000010001111 : data_tmp1 = 10'b0100011010;
13'b000001001000z : data_tmp1 = 10'b0100011010;
13'b000001001001z : data_tmp1 = 10'b0100011011;
13'b000001001010z : data_tmp1 = 10'b0100011100;
13'b0000010010110 : data_tmp1 = 10'b0100011100;
13'b0000010010111 : data_tmp1 = 10'b0100011101;
13'b000001001100z : data_tmp1 = 10'b0100011101;
13'b000001001101z : data_tmp1 = 10'b0100011110;
13'b000001001110z : data_tmp1 = 10'b0100011111;
13'b0000010011110 : data_tmp1 = 10'b0100011111;
13'b0000010011111 : data_tmp1 = 10'b0100100000;
13'b000001010000z : data_tmp1 = 10'b0100100000;
13'b000001010001z : data_tmp1 = 10'b0100100001;
13'b0000010100100 : data_tmp1 = 10'b0100100001;
13'b00000101001z1 : data_tmp1 = 10'b0100100010;
13'b0000010100110 : data_tmp1 = 10'b0100100010;
```

```
13'b000001010100z : data_tmp1 = 10'b0100100011;
13'b0000010101010 : data_tmp1 = 10'b0100100011;

13'b0000010101011 : data_tmp1 = 10'b0100100100;
13'b000001010110z : data_tmp1 = 10'b0100100100;

13'b000001010111z : data_tmp1 = 10'b0100100101;
13'b0000010110000 : data_tmp1 = 10'b0100100101;

13'b00000101100z1 : data_tmp1 = 10'b0100100110;
13'b0000010110010 : data_tmp1 = 10'b0100100110;

13'b000001011010z : data_tmp1 = 10'b0100100111;
13'b0000010110110 : data_tmp1 = 10'b0100100111;

13'b0000010110111 : data_tmp1 = 10'b0100101000;
13'b000001011100z : data_tmp1 = 10'b0100101000;

13'b000001011101z : data_tmp1 = 10'b0100101001;
13'b000001011110z : data_tmp1 = 10'b0100101001;

13'b000001011111z : data_tmp1 = 10'b0100101010;
13'b0000011000000 : data_tmp1 = 10'b0100101010;

13'b00000110000z1 : data_tmp1 = 10'b0100101011;
13'b0000011000010 : data_tmp1 = 10'b0100101011;

13'b00000110001zz : data_tmp1 = 10'b0100101100;

13'b000001100100z : data_tmp1 = 10'b0100101101;
13'b0000011001010 : data_tmp1 = 10'b0100101101;

13'b0000011001011 : data_tmp1 = 10'b0100101110;
13'b000001100110z : data_tmp1 = 10'b0100101110;
13'b0000011001110 : data_tmp1 = 10'b0100101110;

13'b0000011001111 : data_tmp1 = 10'b0100101111;
13'b000001101000z : data_tmp1 = 10'b0100101111;
13'b0000011010010 : data_tmp1 = 10'b0100101111;

13'b0000011010011 : data_tmp1 = 10'b0100110000;
13'b000001101010z : data_tmp1 = 10'b0100110000;

13'b000001101011z : data_tmp1 = 10'b0100110001;
13'b000001101100z : data_tmp1 = 10'b0100110001;

13'b000001101101z : data_tmp1 = 10'b0100110010;
13'b000001101110z : data_tmp1 = 10'b0100110010;

13'b000001101111z : data_tmp1 = 10'b0100110011;
13'b000001110000z : data_tmp1 = 10'b0100110011;

13'b000001110001z : data_tmp1 = 10'b0100110100;
13'b000001110010z : data_tmp1 = 10'b0100110100;

13'b000001110011z : data_tmp1 = 10'b0100110101;
```

```
13'b000001110100z : data_tmp1 = 10'b0100110101;

13'b000001110101z : data_tmp1 = 10'b0100110110;
13'b000001110110z : data_tmp1 = 10'b0100110110;

13'b000001110111z : data_tmp1 = 10'b0100110111;
13'b000001111000z : data_tmp1 = 10'b0100110111;

13'b000001111001z : data_tmp1 = 10'b0100111000;
13'b000001111010z : data_tmp1 = 10'b0100111000;
13'b0000011110110 : data_tmp1 = 10'b0100111000;

13'b0000011110111 : data_tmp1 = 10'b0100111001;
13'b000001111100z : data_tmp1 = 10'b0100111001;
13'b0000011111010 : data_tmp1 = 10'b0100111001;

13'b0000011111011 : data_tmp1 = 10'b0100111010;
13'b000001111110z : data_tmp1 = 10'b0100111010;
13'b0000011111110 : data_tmp1 = 10'b0100111010;

13'b0000011111111 : data_tmp1 = 10'b0100111011;
13'b00001000000zz : data_tmp1 = 10'b0100111011;

13'b00001000001zz : data_tmp1 = 10'b0100111100;
13'b0000100001000 : data_tmp1 = 10'b0100111100;

13'b00001000010z1 : data_tmp1 = 10'b0100111101;
13'b0000100001010 : data_tmp1 = 10'b0100111101;
13'b0000100001100 : data_tmp1 = 10'b0100111101;

13'b00001000011z1 : data_tmp1 = 10'b0100111110;
13'b0000100001110 : data_tmp1 = 10'b0100111110;
13'b000010001000z : data_tmp1 = 10'b0100111110;

13'b000010001001z : data_tmp1 = 10'b0100111111;
13'b000010001010z : data_tmp1 = 10'b0100111111;
13'b0000100010110 : data_tmp1 = 10'b0100111111;

13'b0000100010111 : data_tmp1 = 10'b0101000000;
13'b00001000110zz : data_tmp1 = 10'b0101000000;

13'b00001000111zz : data_tmp1 = 10'b0101000001;
13'b0000100100000 : data_tmp1 = 10'b0101000001;

13'b00001001000z1 : data_tmp1 = 10'b0101000010;
13'b0000100100010 : data_tmp1 = 10'b0101000010;
13'b000010010010z : data_tmp1 = 10'b0101000010;

13'b000010010011z : data_tmp1 = 10'b0101000011;
13'b000010010100z : data_tmp1 = 10'b0101000011;
13'b0000100101010 : data_tmp1 = 10'b0101000011;

13'b0000100101z11 : data_tmp1 = 10'b0101000100;
13'b000010010110z : data_tmp1 = 10'b0101000100;
13'b0000100101110 : data_tmp1 = 10'b0101000100;
13'b0000100110000 : data_tmp1 = 10'b0101000100;
```

```
13'b00001001100z1 : data_tmp1 = 10'b0101000101;
13'b0000100110010 : data_tmp1 = 10'b0101000101;
13'b000010011010z : data_tmp1 = 10'b0101000101;

13'b000010011011z : data_tmp1 = 10'b0101000110;
13'b000010011100z : data_tmp1 = 10'b0101000110;
13'b0000100111010 : data_tmp1 = 10'b0101000110;

13'b0000100111z11 : data_tmp1 = 10'b0101000111;
13'b000010011110z : data_tmp1 = 10'b0101000111;
13'b0000100111110 : data_tmp1 = 10'b0101000111;
13'b0000101000000 : data_tmp1 = 10'b0101000111;

13'b00001010000z1 : data_tmp1 = 10'b0101001000;
13'b0000101000z10 : data_tmp1 = 10'b0101001000;
13'b000010100010z : data_tmp1 = 10'b0101001000;

13'b0000101000111 : data_tmp1 = 10'b0101001001;
13'b00001010010zz : data_tmp1 = 10'b0101001001;
13'b0000101001100 : data_tmp1 = 10'b0101001001;

13'b00001010011z1 : data_tmp1 = 10'b0101001010;
13'b0000101001110 : data_tmp1 = 10'b0101001010;
13'b000010101000z : data_tmp1 = 10'b0101001010;

13'b0000101010z1z : data_tmp1 = 10'b0101001011;
13'b000010101010z : data_tmp1 = 10'b0101001011;

13'b00001010110zz : data_tmp1 = 10'b0101001100;
13'b000010101110z : data_tmp1 = 10'b0101001100;
13'b0000101011110 : data_tmp1 = 10'b0101001100;

13'b0000101011111 : data_tmp1 = 10'b0101001101;
13'b00001011000zz : data_tmp1 = 10'b0101001101;
13'b0000101100100 : data_tmp1 = 10'b0101001101;

13'b00001011001z1 : data_tmp1 = 10'b0101001110;
13'b0000101100110 : data_tmp1 = 10'b0101001110;
13'b000010110100z : data_tmp1 = 10'b0101001110;
13'b0000101101010 : data_tmp1 = 10'b0101001110;

13'b0000101101z11 : data_tmp1 = 10'b0101001111;
13'b000010110110z : data_tmp1 = 10'b0101001111;
13'b0000101101110 : data_tmp1 = 10'b0101001111;
13'b0000101110000 : data_tmp1 = 10'b0101001111;

13'b00001011100z1 : data_tmp1 = 10'b0101010000;
13'b0000101110z10 : data_tmp1 = 10'b0101010000;
13'b000010111010z : data_tmp1 = 10'b0101010000;
13'b0000101110111 : data_tmp1 = 10'b0101010000;

13'b00001011110zz : data_tmp1 = 10'b0101010001;
13'b000010111110z : data_tmp1 = 10'b0101010001;
13'b0000101111110 : data_tmp1 = 10'b0101010001;

13'b0000101111111 : data_tmp1 = 10'b0101010010;
```

```
13'b00001100000zz : data_tmp1 = 10'b0101010010;
13'b000011000010z : data_tmp1 = 10'b0101010010;

13'b000011000011z : data_tmp1 = 10'b0101010011;
13'b00001100010zz : data_tmp1 = 10'b0101010011;

13'b00001100011zz : data_tmp1 = 10'b0101010100;
13'b000011001000z : data_tmp1 = 10'b0101010100;
13'b0000110010010 : data_tmp1 = 10'b0101010100;

13'b0000110010z11 : data_tmp1 = 10'b0101010101;
13'b000011001010z : data_tmp1 = 10'b0101010101;
13'b0000110010110 : data_tmp1 = 10'b0101010101;
13'b000011001100z : data_tmp1 = 10'b0101010101;
13'b0000110011010 : data_tmp1 = 10'b0101010101;

13'b0000110011z11 : data_tmp1 = 10'b0101010110;
13'b000011001110z : data_tmp1 = 10'b0101010110;
13'b0000110011110 : data_tmp1 = 10'b0101010110;
13'b000011010000z : data_tmp1 = 10'b0101010110;

13'b0000110100z1z : data_tmp1 = 10'b0101010111;
13'b000011010010z : data_tmp1 = 10'b0101010111;
13'b0000110101000 : data_tmp1 = 10'b0101010111;

13'b00001101010z1 : data_tmp1 = 10'b0101011000;
13'b0000110101z10 : data_tmp1 = 10'b0101011000;
13'b000011010110z : data_tmp1 = 10'b0101011000;
13'b0000110101111 : data_tmp1 = 10'b0101011000;
13'b0000110110000 : data_tmp1 = 10'b0101011000;

13'b00001101100z1 : data_tmp1 = 10'b0101011001;
13'b0000110110z10 : data_tmp1 = 10'b0101011001;
13'b000011011010z : data_tmp1 = 10'b0101011001;
13'b0000110110111 : data_tmp1 = 10'b0101011001;
13'b0000110111000 : data_tmp1 = 10'b0101011001;

13'b00001101110z1 : data_tmp1 = 10'b0101011010;
13'b0000110111z10 : data_tmp1 = 10'b0101011010;
13'b000011011110z : data_tmp1 = 10'b0101011010;
13'b0000110111111 : data_tmp1 = 10'b0101011010;

13'b0000111000zzz : data_tmp1 = 10'b0101011011;

13'b0000111001zzz : data_tmp1 = 10'b0101011100;

13'b0000111010zzz : data_tmp1 = 10'b0101011101;
13'b0000111011000 : data_tmp1 = 10'b0101011101;

13'b00001110110z1 : data_tmp1 = 10'b0101011110;
13'b0000111011z10 : data_tmp1 = 10'b0101011110;
13'b000011101110z : data_tmp1 = 10'b0101011110;
13'b0000111011111 : data_tmp1 = 10'b0101011110;
13'b0000111100000 : data_tmp1 = 10'b0101011110;

13'b00001111000z1 : data_tmp1 = 10'b0101011111;
```

```
13'b0000111100z10 : data_tmp1 = 10'b0101011111;
13'b000011110010z : data_tmp1 = 10'b0101011111;
13'b0000111100111 : data_tmp1 = 10'b0101011111;
13'b000011110100z : data_tmp1 = 10'b0101011111;

13'b0000111101z1z : data_tmp1 = 10'b0101100000;
13'b000011110110z : data_tmp1 = 10'b0101100000;
13'b000011111000z : data_tmp1 = 10'b0101100000;

13'b0000111110z1z : data_tmp1 = 10'b0101100001;
13'b000011111010z : data_tmp1 = 10'b0101100001;
13'b000011111100z : data_tmp1 = 10'b0101100001;
13'b0000111111010 : data_tmp1 = 10'b0101100001;

13'b0000111111z11 : data_tmp1 = 10'b0101100010;
13'b000011111110z : data_tmp1 = 10'b0101100010;
13'b0000111111110 : data_tmp1 = 10'b0101100010;
13'b00010000000zz : data_tmp1 = 10'b0101100010;

13'b00010000001zz : data_tmp1 = 10'b0101100011;
13'b00010000010zz : data_tmp1 = 10'b0101100011;
13'b0001000001100 : data_tmp1 = 10'b0101100011;

13'b00010000011z1 : data_tmp1 = 10'b0101100100;
13'b0001000001110 : data_tmp1 = 10'b0101100100;
13'b00010000100zz : data_tmp1 = 10'b0101100100;
13'b000100001010z : data_tmp1 = 10'b0101100100;
13'b0001000010110 : data_tmp1 = 10'b0101100100;

13'b000100001z111 : data_tmp1 = 10'b0101100101;
13'b00010000110zz : data_tmp1 = 10'b0101100101;
13'b000100001110z : data_tmp1 = 10'b0101100101;
13'b0001000011110 : data_tmp1 = 10'b0101100101;

13'b0001000100zzz : data_tmp1 = 10'b0101100110;
13'b000100010100z : data_tmp1 = 10'b0101100110;

13'b0001000101z1z : data_tmp1 = 10'b0101100111;
13'b000100010110z : data_tmp1 = 10'b0101100111;
13'b00010001100zz : data_tmp1 = 10'b0101100111;

13'b00010001101zz : data_tmp1 = 10'b0101101000;
13'b00010001110zz : data_tmp1 = 10'b0101101000;
13'b000100011110z : data_tmp1 = 10'b0101101000;

13'b000100011111z : data_tmp1 = 10'b0101101001;
13'b0001001000zzz : data_tmp1 = 10'b0101101001;

13'b0001001001zzz : data_tmp1 = 10'b0101101010;
13'b000100101000z : data_tmp1 = 10'b0101101010;

13'b0001001010z1z : data_tmp1 = 10'b0101101011;
13'b000100101010z : data_tmp1 = 10'b0101101011;
13'b00010010110zz : data_tmp1 = 10'b0101101011;
13'b0001001011100 : data_tmp1 = 10'b0101101011;
```

```
13'b00010010111z1 : data_tmp1 = 10'b0101101100;
13'b0001001011110 : data_tmp1 = 10'b0101101100;
13'b00010011000zz : data_tmp1 = 10'b0101101100;
13'b000100110010z : data_tmp1 = 10'b0101101100;
13'b0001001100110 : data_tmp1 = 10'b0101101100;

13'b000100110z111 : data_tmp1 = 10'b0101101101;
13'b00010011010zz : data_tmp1 = 10'b0101101101;
13'b000100110110z : data_tmp1 = 10'b0101101101;
13'b0001001101110 : data_tmp1 = 10'b0101101101;
13'b000100111000z : data_tmp1 = 10'b0101101101;

13'b0001001110z1z : data_tmp1 = 10'b0101101110;
13'b000100111010z : data_tmp1 = 10'b0101101110;
13'b00010011110zz : data_tmp1 = 10'b0101101110;
13'b0001001111100 : data_tmp1 = 10'b0101101110;

13'b00010011111z1 : data_tmp1 = 10'b0101101111;
13'b0001001111110 : data_tmp1 = 10'b0101101111;
13'b0001010000zzz : data_tmp1 = 10'b0101101111;
13'b0001010001000 : data_tmp1 = 10'b0101101111;

13'b00010100010z1 : data_tmp1 = 10'b0101110000;
13'b0001010001z10 : data_tmp1 = 10'b0101110000;
13'b000101000110z : data_tmp1 = 10'b0101110000;
13'b0001010001111 : data_tmp1 = 10'b0101110000;
13'b00010100100zz : data_tmp1 = 10'b0101110000;

13'b000101001z1zz : data_tmp1 = 10'b0101110001;
13'b00010100110zz : data_tmp1 = 10'b0101110001;

13'b0001010100zzz : data_tmp1 = 10'b0101110010;
13'b00010101010zz : data_tmp1 = 10'b0101110010;

13'b00010101011zz : data_tmp1 = 10'b0101110011;
13'b0001010110zzz : data_tmp1 = 10'b0101110011;

13'b0001010111zzz : data_tmp1 = 10'b0101110100;
13'b00010110000zz : data_tmp1 = 10'b0101110100;

13'b000101100z1zz : data_tmp1 = 10'b0101110101;
13'b00010110010zz : data_tmp1 = 10'b0101110101;
13'b0001011010000 : data_tmp1 = 10'b0101110101;

13'b00010110100z1 : data_tmp1 = 10'b0101110110;
13'b0001011010z10 : data_tmp1 = 10'b0101110110;
13'b000101101z10z : data_tmp1 = 10'b0101110110;
13'b0001011010111 : data_tmp1 = 10'b0101110110;
13'b00010110110zz : data_tmp1 = 10'b0101110110;

13'b000101101111z : data_tmp1 = 10'b0101110111;
13'b0001011100zzz : data_tmp1 = 10'b0101110111;
13'b000101110100z : data_tmp1 = 10'b0101110111;
13'b0001011101010 : data_tmp1 = 10'b0101110111;

13'b0001011101z11 : data_tmp1 = 10'b0101111000;
```

```
13'b000101110110z : data_tmp1 = 10'b0101111000;
13'b0001011101110 : data_tmp1 = 10'b0101111000;
13'b0001011110zzz : data_tmp1 = 10'b0101111000;

13'b0001011111zzz : data_tmp1 = 10'b0101111001;
13'b00011000000zz : data_tmp1 = 10'b0101111001;
13'b0001100000100 : data_tmp1 = 10'b0101111001;

13'b00011000001z1 : data_tmp1 = 10'b0101111010;
13'b000110000z110 : data_tmp1 = 10'b0101111010;
13'b00011000010zz : data_tmp1 = 10'b0101111010;
13'b000110000110z : data_tmp1 = 10'b0101111010;
13'b0001100001111 : data_tmp1 = 10'b0101111010;
13'b000110001000z : data_tmp1 = 10'b0101111010;
13'b0001100010010 : data_tmp1 = 10'b0101111010;

13'b0001100010z11 : data_tmp1 = 10'b0101111011;
13'b000110001z10z : data_tmp1 = 10'b0101111011;
13'b000110001z110 : data_tmp1 = 10'b0101111011;
13'b00011000110zz : data_tmp1 = 10'b0101111011;
13'b0001100011111 : data_tmp1 = 10'b0101111011;
13'b0001100100000 : data_tmp1 = 10'b0101111011;

13'b00011001000z1 : data_tmp1 = 10'b0101111100;
13'b0001100100z10 : data_tmp1 = 10'b0101111100;
13'b000110010z10z : data_tmp1 = 10'b0101111100;
13'b0001100100111 : data_tmp1 = 10'b0101111100;
13'b00011001010zz : data_tmp1 = 10'b0101111100;
13'b0001100101110 : data_tmp1 = 10'b0101111100;

13'b0001100101111 : data_tmp1 = 10'b0101111101;
13'b0001100110zzz : data_tmp1 = 10'b0101111101;
13'b00011001110zz : data_tmp1 = 10'b0101111101;
13'b000110011110z : data_tmp1 = 10'b0101111101;

13'b000110011111z : data_tmp1 = 10'b0101111110;
13'b0001101000zzz : data_tmp1 = 10'b0101111110;
13'b00011010010zz : data_tmp1 = 10'b0101111110;
13'b0001101001100 : data_tmp1 = 10'b0101111110;

13'b00011010011z1 : data_tmp1 = 10'b0101111111;
13'b0001101001110 : data_tmp1 = 10'b0101111111;
13'b0001101010zzz : data_tmp1 = 10'b0101111111;
13'b00011010110zz : data_tmp1 = 10'b0101111111;

13'b00011010111zz : data_tmp1 = 10'b0110000000;
13'b0001101100zzz : data_tmp1 = 10'b0110000000;
13'b000110110100z : data_tmp1 = 10'b0110000000;
13'b0001101101010 : data_tmp1 = 10'b0110000000;

13'b0001101101z11 : data_tmp1 = 10'b0110000001;
13'b000110110110z : data_tmp1 = 10'b0110000001;
13'b0001101101110 : data_tmp1 = 10'b0110000001;
13'b0001101110zzz : data_tmp1 = 10'b0110000001;
13'b000110111100z : data_tmp1 = 10'b0110000001;
```

```
13'b0001101111z1z : data_tmp1 = 10'b0110000010;
13'b000110111110z : data_tmp1 = 10'b0110000010;
13'b0001110000zzz : data_tmp1 = 10'b0110000010;
13'b000111000100z : data_tmp1 = 10'b0110000010;

13'b0001110001z1z : data_tmp1 = 10'b0110000011;
13'b000111000110z : data_tmp1 = 10'b0110000011;
13'b0001110010zzz : data_tmp1 = 10'b0110000011;
13'b000111001100z : data_tmp1 = 10'b0110000011;

13'b0001110011z1z : data_tmp1 = 10'b0110000100;
13'b000111001110z : data_tmp1 = 10'b0110000100;
13'b0001110100zzz : data_tmp1 = 10'b0110000100;
13'b000111010100z : data_tmp1 = 10'b0110000100;
13'b0001110101010 : data_tmp1 = 10'b0110000100;

13'b0001110101z11 : data_tmp1 = 10'b0110000101;
13'b000111010110z : data_tmp1 = 10'b0110000101;
13'b0001110101110 : data_tmp1 = 10'b0110000101;
13'b0001110110zzz : data_tmp1 = 10'b0110000101;
13'b000111011100z : data_tmp1 = 10'b0110000101;
13'b0001110111010 : data_tmp1 = 10'b0110000101;

13'b0001110111z11 : data_tmp1 = 10'b0110000110;
13'b000111011110z : data_tmp1 = 10'b0110000110;
13'b0001110111110 : data_tmp1 = 10'b0110000110;
13'b0001111000zzz : data_tmp1 = 10'b0110000110;
13'b00011110010zz : data_tmp1 = 10'b0110000110;

13'b00011110011zz : data_tmp1 = 10'b0110000111;
13'b0001111010zzz : data_tmp1 = 10'b0110000111;
13'b00011110110zz : data_tmp1 = 10'b0110000111;
13'b0001111011100 : data_tmp1 = 10'b0110000111;

13'b00011110111z1 : data_tmp1 = 10'b0110001000;
13'b0001111011110 : data_tmp1 = 10'b0110001000;
13'b0001111100zzz : data_tmp1 = 10'b0110001000;
13'b00011111010zz : data_tmp1 = 10'b0110001000;
13'b000111110110z : data_tmp1 = 10'b0110001000;
13'b0001111101110 : data_tmp1 = 10'b0110001000;

13'b00011111z1111 : data_tmp1 = 10'b0110001001;
13'b0001111110zzz : data_tmp1 = 10'b0110001001;
13'b00011111110zz : data_tmp1 = 10'b0110001001;
13'b000111111110z : data_tmp1 = 10'b0110001001;
13'b0001111111110 : data_tmp1 = 10'b0110001001;
13'b0010000000000 : data_tmp1 = 10'b0110001001;

13'b00100000000z1 : data_tmp1 = 10'b0110001010;
13'b0010000000z10 : data_tmp1 = 10'b0110001010;
13'b001000000z10z : data_tmp1 = 10'b0110001010;
13'b001000000z111 : data_tmp1 = 10'b0110001010;
13'b00100000010zz : data_tmp1 = 10'b0110001010;
13'b0010000001110 : data_tmp1 = 10'b0110001010;
13'b001000001000z : data_tmp1 = 10'b0110001010;
13'b0010000010010 : data_tmp1 = 10'b0110001010;
```

```
13'b0010000010z11 : data_tmp1 = 10'b0110001011;
13'b001000001z10z : data_tmp1 = 10'b0110001011;
13'b001000001z110 : data_tmp1 = 10'b0110001011;
13'b00100000110zz : data_tmp1 = 10'b0110001011;
13'b0010000011111 : data_tmp1 = 10'b0110001011;
13'b00100001000zz : data_tmp1 = 10'b0110001011;
13'b001000010010z : data_tmp1 = 10'b0110001011;

13'b001000010z11z : data_tmp1 = 10'b0110001100;
13'b00100001010zz : data_tmp1 = 10'b0110001100;
13'b001000010110z : data_tmp1 = 10'b0110001100;
13'b0010000110zzz : data_tmp1 = 10'b0110001100;

13'b0010000111zzz : data_tmp1 = 10'b0110001101;
13'b0010001000zzz : data_tmp1 = 10'b0110001101;
13'b00100010010zz : data_tmp1 = 10'b0110001101;

13'b00100010011zz : data_tmp1 = 10'b0110001110;
13'b0010001010zzz : data_tmp1 = 10'b0110001110;
13'b00100010110zz : data_tmp1 = 10'b0110001110;
13'b001000101110z : data_tmp1 = 10'b0110001110;
13'b0010001011110 : data_tmp1 = 10'b0110001110;

13'b0010001011111 : data_tmp1 = 10'b0110001111;
13'b001000110zzzz : data_tmp1 = 10'b0110001111;
13'b001000111000z : data_tmp1 = 10'b0110001111;
13'b0010001110010 : data_tmp1 = 10'b0110001111;

13'b0010001110z11 : data_tmp1 = 10'b0110010000;
13'b001000111z10z : data_tmp1 = 10'b0110010000;
13'b001000111z110 : data_tmp1 = 10'b0110010000;
13'b00100011110zz : data_tmp1 = 10'b0110010000;
13'b0010001111111 : data_tmp1 = 10'b0110010000;
13'b00100100000zz : data_tmp1 = 10'b0110010000;
13'b001001000010z : data_tmp1 = 10'b0110010000;
13'b0010010000110 : data_tmp1 = 10'b0110010000;

13'b001001000z111 : data_tmp1 = 10'b0110010001;
13'b00100100z10zz : data_tmp1 = 10'b0110010001;
13'b001001000110z : data_tmp1 = 10'b0110010001;
13'b0010010001110 : data_tmp1 = 10'b0110010001;
13'b0010010010zzz : data_tmp1 = 10'b0110010001;

13'b00100100111zz : data_tmp1 = 10'b0110010010;
13'b001001010zzzz : data_tmp1 = 10'b0110010010;
13'b0010010110000 : data_tmp1 = 10'b0110010010;

13'b00100101100z1 : data_tmp1 = 10'b0110010011;
13'b0010010110z10 : data_tmp1 = 10'b0110010011;
13'b001001011z10z : data_tmp1 = 10'b0110010011;
13'b001001011z111 : data_tmp1 = 10'b0110010011;
13'b00100101110zz : data_tmp1 = 10'b0110010011;
13'b0010010111110 : data_tmp1 = 10'b0110010011;
13'b00100110000zz : data_tmp1 = 10'b0110010011;
13'b001001100010z : data_tmp1 = 10'b0110010011;
```

```
13'b001001100z11z : data_tmp1 = 10'b0110010100;
13'b00100110z10zz : data_tmp1 = 10'b0110010100;
13'b001001100110z : data_tmp1 = 10'b0110010100;
13'b0010011010zzz : data_tmp1 = 10'b0110010100;

13'b00100110111zz : data_tmp1 = 10'b0110010101;
13'b001001110zzzz : data_tmp1 = 10'b0110010101;
13'b001001111000z : data_tmp1 = 10'b0110010101;

13'b0010011110z1z : data_tmp1 = 10'b0110010110;
13'b001001111z10z : data_tmp1 = 10'b0110010110;
13'b00100111110zz : data_tmp1 = 10'b0110010110;
13'b001001111111z : data_tmp1 = 10'b0110010110;
13'b0010100000zzz : data_tmp1 = 10'b0110010110;

13'b0010100001zzz : data_tmp1 = 10'b0110010111;
13'b0010100010zzz : data_tmp1 = 10'b0110010111;
13'b00101000110zz : data_tmp1 = 10'b0110010111;
13'b001010001110z : data_tmp1 = 10'b0110010111;
13'b0010100011110 : data_tmp1 = 10'b0110010111;

13'b0010100011111 : data_tmp1 = 10'b0110011000;
13'b001010010zzzz : data_tmp1 = 10'b0110011000;
13'b00101001100zz : data_tmp1 = 10'b0110011000;
13'b001010011010z : data_tmp1 = 10'b0110011000;

13'b001010011z11z : data_tmp1 = 10'b0110011001;
13'b00101001110zz : data_tmp1 = 10'b0110011001;
13'b001010011110z : data_tmp1 = 10'b0110011001;
13'b0010101000zzz : data_tmp1 = 10'b0110011001;
13'b00101010010zz : data_tmp1 = 10'b0110011001;
13'b001010100110z : data_tmp1 = 10'b0110011001;

13'b00101010z111z : data_tmp1 = 10'b0110011010;
13'b0010101010zzz : data_tmp1 = 10'b0110011010;
13'b00101010110zz : data_tmp1 = 10'b0110011010;
13'b001010101110z : data_tmp1 = 10'b0110011010;
13'b00101011000zz : data_tmp1 = 10'b0110011010;
13'b001010110010z : data_tmp1 = 10'b0110011010;

13'b001010110z11z : data_tmp1 = 10'b0110011011;
13'b00101011z10zz : data_tmp1 = 10'b0110011011;
13'b00101011z110z : data_tmp1 = 10'b0110011011;
13'b0010101110zzz : data_tmp1 = 10'b0110011011;
13'b0010101111110 : data_tmp1 = 10'b0110011011;

13'b0010101111111 : data_tmp1 = 10'b0110011100;
13'b001011000zzzz : data_tmp1 = 10'b0110011100;
13'b0010110010zzz : data_tmp1 = 10'b0110011100;

13'b0010110011zzz : data_tmp1 = 10'b0110011101;
13'b001011010zzzz : data_tmp1 = 10'b0110011101;
13'b0010110110000 : data_tmp1 = 10'b0110011101;

13'b00101101100z1 : data_tmp1 = 10'b0110011110;
13'b0010110110z10 : data_tmp1 = 10'b0110011110;
```

```
13'b001011011z10z : data_tmp1 = 10'b0110011110;
13'b001011011z111 : data_tmp1 = 10'b0110011110;
13'b00101101110zz : data_tmp1 = 10'b0110011110;
13'b0010110111110 : data_tmp1 = 10'b0110011110;
13'b0010111000zzz : data_tmp1 = 10'b0110011110;
13'b001011100100z : data_tmp1 = 10'b0110011110;
13'b0010111001010 : data_tmp1 = 10'b0110011110;

13'b0010111001z11 : data_tmp1 = 10'b0110011111;
13'b00101110z110z : data_tmp1 = 10'b0110011111;
13'b00101110z1110 : data_tmp1 = 10'b0110011111;
13'b0010111010zzz : data_tmp1 = 10'b0110011111;
13'b00101110110zz : data_tmp1 = 10'b0110011111;
13'b0010111011111 : data_tmp1 = 10'b0110011111;
13'b00101111000zz : data_tmp1 = 10'b0110011111;
13'b0010111100100 : data_tmp1 = 10'b0110011111;

13'b00101111001z1 : data_tmp1 = 10'b0110100000;
13'b001011110z110 : data_tmp1 = 10'b0110100000;
13'b00101111z10zz : data_tmp1 = 10'b0110100000;
13'b00101111z110z : data_tmp1 = 10'b0110100000;
13'b00101111z1111 : data_tmp1 = 10'b0110100000;
13'b0010111110zzz : data_tmp1 = 10'b0110100000;
13'b0010111111110 : data_tmp1 = 10'b0110100000;

13'b001100000zzzz : data_tmp1 = 10'b0110100001;
13'b0011000010zzz : data_tmp1 = 10'b0110100001;
13'b00110000110zz : data_tmp1 = 10'b0110100001;

13'b00110000111zz : data_tmp1 = 10'b0110100010;
13'b001100010zzzz : data_tmp1 = 10'b0110100010;
13'b00110001100zz : data_tmp1 = 10'b0110100010;
13'b001100011010z : data_tmp1 = 10'b0110100010;
13'b0011000110110 : data_tmp1 = 10'b0110100010;

13'b001100011z111 : data_tmp1 = 10'b0110100011;
13'b00110001110zz : data_tmp1 = 10'b0110100011;
13'b001100011110z : data_tmp1 = 10'b0110100011;
13'b0011000111110 : data_tmp1 = 10'b0110100011;
13'b001100100zzzz : data_tmp1 = 10'b0110100011;
13'b00110010100zz : data_tmp1 = 10'b0110100011;

13'b001100101z1zz : data_tmp1 = 10'b0110100100;
13'b00110010110zz : data_tmp1 = 10'b0110100100;
13'b001100110zzzz : data_tmp1 = 10'b0110100100;

13'b001100111zzzz : data_tmp1 = 10'b0110100101;
13'b0011010000zzz : data_tmp1 = 10'b0110100101;
13'b00110100010zz : data_tmp1 = 10'b0110100101;
13'b001101000110z : data_tmp1 = 10'b0110100101;

13'b00110100z111z : data_tmp1 = 10'b0110100110;
13'b0011010010zzz : data_tmp1 = 10'b0110100110;
13'b00110100110zz : data_tmp1 = 10'b0110100110;
13'b001101001110z : data_tmp1 = 10'b0110100110;
13'b0011010100zzz : data_tmp1 = 10'b0110100110;
```

```
13'b001101010100z : data_tmp1 = 10'b0110100110;
13'b0011010101010 : data_tmp1 = 10'b0110100110;

13'b0011010101z11 : data_tmp1 = 10'b0110100111;
13'b00110101z110z : data_tmp1 = 10'b0110100111;
13'b00110101z1110 : data_tmp1 = 10'b0110100111;
13'b0011010110zzz : data_tmp1 = 10'b0110100111;
13'b00110101110zz : data_tmp1 = 10'b0110100111;
13'b0011010111111 : data_tmp1 = 10'b0110100111;
13'b0011011000zzz : data_tmp1 = 10'b0110100111;
13'b001101100100z : data_tmp1 = 10'b0110100111;

13'b0011011001z1z : data_tmp1 = 10'b0110101000;
13'b00110110z110z : data_tmp1 = 10'b0110101000;
13'b0011011010zzz : data_tmp1 = 10'b0110101000;
13'b00110110110zz : data_tmp1 = 10'b0110101000;
13'b001101101111z : data_tmp1 = 10'b0110101000;
13'b0011011100zzz : data_tmp1 = 10'b0110101000;
13'b0011011101000 : data_tmp1 = 10'b0110101000;

13'b00110111010z1 : data_tmp1 = 10'b0110101001;
13'b0011011101z10 : data_tmp1 = 10'b0110101001;
13'b00110111z110z : data_tmp1 = 10'b0110101001;
13'b00110111z1111 : data_tmp1 = 10'b0110101001;
13'b0011011110zzz : data_tmp1 = 10'b0110101001;
13'b00110111110zz : data_tmp1 = 10'b0110101001;
13'b0011011111110 : data_tmp1 = 10'b0110101001;
13'b0011100000zzz : data_tmp1 = 10'b0110101001;

13'b00111000z1zzz : data_tmp1 = 10'b0110101010;
13'b0011100010zzz : data_tmp1 = 10'b0110101010;
13'b0011100100zzz : data_tmp1 = 10'b0110101010;

13'b00111001z1zzz : data_tmp1 = 10'b0110101011;
13'b0011100110zzz : data_tmp1 = 10'b0110101011;
13'b0011101000zzz : data_tmp1 = 10'b0110101011;

13'b00111010z1zzz : data_tmp1 = 10'b0110101100;
13'b0011101010zzz : data_tmp1 = 10'b0110101100;
13'b0011101100zzz : data_tmp1 = 10'b0110101100;
13'b0011101101000 : data_tmp1 = 10'b0110101100;

13'b00111011010z1 : data_tmp1 = 10'b0110101101;
13'b0011101101z10 : data_tmp1 = 10'b0110101101;
13'b00111011z110z : data_tmp1 = 10'b0110101101;
13'b00111011z1111 : data_tmp1 = 10'b0110101101;
13'b0011101110zzz : data_tmp1 = 10'b0110101101;
13'b00111011110zz : data_tmp1 = 10'b0110101101;
13'b0011101111110 : data_tmp1 = 10'b0110101101;
13'b0011110000zzz : data_tmp1 = 10'b0110101101;
13'b001111000100z : data_tmp1 = 10'b0110101101;
13'b0011110001010 : data_tmp1 = 10'b0110101101;

13'b0011110001z11 : data_tmp1 = 10'b0110101110;
13'b00111100z110z : data_tmp1 = 10'b0110101110;
13'b00111100z1110 : data_tmp1 = 10'b0110101110;
```

```
13'b0011110010zzz : data_tmp1 = 10'b0110101110;
13'b00111100110zz : data_tmp1 = 10'b0110101110;
13'b0011110011111 : data_tmp1 = 10'b0110101110;
13'b0011110100zzz : data_tmp1 = 10'b0110101110;
13'b00111101010zz : data_tmp1 = 10'b0110101110;
13'b0011110101100 : data_tmp1 = 10'b0110101110;

13'b00111101011z1 : data_tmp1 = 10'b0110101111;
13'b00111101z1110 : data_tmp1 = 10'b0110101111;
13'b0011110110zzz : data_tmp1 = 10'b0110101111;
13'b00111101110zz : data_tmp1 = 10'b0110101111;
13'b001111011110z : data_tmp1 = 10'b0110101111;
13'b0011110111111 : data_tmp1 = 10'b0110101111;
13'b001111100zzzz : data_tmp1 = 10'b0110101111;

13'b001111101zzzz : data_tmp1 = 10'b0110110000;
13'b001111110zzzz : data_tmp1 = 10'b0110110000;
13'b00111111100zz : data_tmp1 = 10'b0110110000;

13'b001111111z1zz : data_tmp1 = 10'b0110110001;
13'b00111111110zz : data_tmp1 = 10'b0110110001;
13'b010000000zzzz : data_tmp1 = 10'b0110110001;
13'b0100000010zzz : data_tmp1 = 10'b0110110001;

13'b0100000011zzz : data_tmp1 = 10'b0110110010;
13'b010000010zzzz : data_tmp1 = 10'b0110110010;
13'b0100000110zzz : data_tmp1 = 10'b0110110010;
13'b01000001110zz : data_tmp1 = 10'b0110110010;
13'b0100000111100 : data_tmp1 = 10'b0110110010;

13'b01000001111z1 : data_tmp1 = 10'b0110110011;
13'b0100000111110 : data_tmp1 = 10'b0110110011;
13'b01000010zzzzz : data_tmp1 = 10'b0110110011;
13'b010000110000z : data_tmp1 = 10'b0110110011;

13'b0100001100z1z : data_tmp1 = 10'b0110110100;
13'b010000110z10z : data_tmp1 = 10'b0110110100;
13'b01000011z10zz : data_tmp1 = 10'b0110110100;
13'b01000011z111z : data_tmp1 = 10'b0110110100;
13'b0100001110zzz : data_tmp1 = 10'b0110110100;
13'b010000111110z : data_tmp1 = 10'b0110110100;
13'b0100010000zzz : data_tmp1 = 10'b0110110100;

13'b01000100z1zzz : data_tmp1 = 10'b0110110101;
13'b0100010010zzz : data_tmp1 = 10'b0110110101;
13'b0100010100zzz : data_tmp1 = 10'b0110110101;
13'b01000101010zz : data_tmp1 = 10'b0110110101;
13'b010001010110z : data_tmp1 = 10'b0110110101;
13'b0100010101110 : data_tmp1 = 10'b0110110101;

13'b01000101z1111 : data_tmp1 = 10'b0110110110;
13'b0100010110zzz : data_tmp1 = 10'b0110110110;
13'b01000101110zz : data_tmp1 = 10'b0110110110;
13'b010001011110z : data_tmp1 = 10'b0110110110;
13'b0100010111110 : data_tmp1 = 10'b0110110110;
13'b010001100zzzz : data_tmp1 = 10'b0110110110;
```

```
13'b01000110100zz : data_tmp1 = 10'b0110110110;
13'b010001101010z : data_tmp1 = 10'b0110110110;
13'b0100011010110 : data_tmp1 = 10'b0110110110;

13'b010001101z111 : data_tmp1 = 10'b0110110111;
13'b0100011z110zz : data_tmp1 = 10'b0110110111;
13'b0100011z1110z : data_tmp1 = 10'b0110110111;
13'b0100011z11110 : data_tmp1 = 10'b0110110111;
13'b010001110zzzz : data_tmp1 = 10'b0110110111;
13'b0100011110zzz : data_tmp1 = 10'b0110110111;

13'b0100011111111 : data_tmp1 = 10'b0110111000;
13'b01001000zzzzz : data_tmp1 = 10'b0110111000;
13'b0100100100zzz : data_tmp1 = 10'b0110111000;

13'b01001001z1zzz : data_tmp1 = 10'b0110111001;
13'b0100100110zzz : data_tmp1 = 10'b0110111001;
13'b010010100zzzz : data_tmp1 = 10'b0110111001;
13'b0100101010000 : data_tmp1 = 10'b0110111001;

13'b01001010100z1 : data_tmp1 = 10'b0110111010;
13'b0100101010z10 : data_tmp1 = 10'b0110111010;
13'b010010101z10z : data_tmp1 = 10'b0110111010;
13'b010010101z111 : data_tmp1 = 10'b0110111010;
13'b0100101z110zz : data_tmp1 = 10'b0110111010;
13'b0100101011110 : data_tmp1 = 10'b0110111010;
13'b010010110zzzz : data_tmp1 = 10'b0110111010;
13'b0100101110zzz : data_tmp1 = 10'b0110111010;

13'b01001011111zz : data_tmp1 = 10'b0110111011;
13'b01001100zzzzz : data_tmp1 = 10'b0110111011;
13'b01001101000zz : data_tmp1 = 10'b0110111011;
13'b010011010010z : data_tmp1 = 10'b0110111011;
13'b0100110100110 : data_tmp1 = 10'b0110111011;

13'b010011010z111 : data_tmp1 = 10'b0110111100;
13'b01001101z10zz : data_tmp1 = 10'b0110111100;
13'b01001101z110z : data_tmp1 = 10'b0110111100;
13'b01001101z1110 : data_tmp1 = 10'b0110111100;
13'b0100110110zzz : data_tmp1 = 10'b0110111100;
13'b0100110111111 : data_tmp1 = 10'b0110111100;
13'b010011100zzzz : data_tmp1 = 10'b0110111100;
13'b010011101000z : data_tmp1 = 10'b0110111100;
13'b0100111010010 : data_tmp1 = 10'b0110111100;

13'b0100111010z11 : data_tmp1 = 10'b0110111101;
13'b010011101z10z : data_tmp1 = 10'b0110111101;
13'b010011101z110 : data_tmp1 = 10'b0110111101;
13'b0100111z110zz : data_tmp1 = 10'b0110111101;
13'b0100111011111 : data_tmp1 = 10'b0110111101;
13'b010011110zzzz : data_tmp1 = 10'b0110111101;
13'b0100111110zzz : data_tmp1 = 10'b0110111101;
13'b010011111110z : data_tmp1 = 10'b0110111101;
13'b0100111111110 : data_tmp1 = 10'b0110111101;

13'b0100111111111 : data_tmp1 = 10'b0110111110;
```

```
13'b01010000zzzzz : data_tmp1 = 10'b0110111110;
13'b0101000100zzz : data_tmp1 = 10'b0110111110;
13'b01010001010zz : data_tmp1 = 10'b0110111110;
13'b0101000101100 : data_tmp1 = 10'b0110111110;

13'b01010001011z1 : data_tmp1 = 10'b0110111111;
13'b01010001z1110 : data_tmp1 = 10'b0110111111;
13'b0101000110zzz : data_tmp1 = 10'b0110111111;
13'b01010001110zz : data_tmp1 = 10'b0110111111;
13'b010100011110z : data_tmp1 = 10'b0110111111;
13'b0101000111111 : data_tmp1 = 10'b0110111111;
13'b010100100zzzz : data_tmp1 = 10'b0110111111;
13'b0101001010zzz : data_tmp1 = 10'b0110111111;
13'b010100101100z : data_tmp1 = 10'b0110111111;
13'b0101001011010 : data_tmp1 = 10'b0110111111;

13'b0101001011z11 : data_tmp1 = 10'b0111000000;
13'b0101001z1110z : data_tmp1 = 10'b0111000000;
13'b0101001z11110 : data_tmp1 = 10'b0111000000;
13'b010100110zzzz : data_tmp1 = 10'b0111000000;
13'b0101001110zzz : data_tmp1 = 10'b0111000000;
13'b01010011110zz : data_tmp1 = 10'b0111000000;
13'b0101001111111 : data_tmp1 = 10'b0111000000;
13'b0101010000zzz : data_tmp1 = 10'b0111000000;
13'b010101000100z : data_tmp1 = 10'b0111000000;

13'b0101010001z1z : data_tmp1 = 10'b0111000001;
13'b01010100z110z : data_tmp1 = 10'b0111000001;
13'b0101010z10zzz : data_tmp1 = 10'b0111000001;
13'b01010100110zz : data_tmp1 = 10'b0111000001;
13'b010101001111z : data_tmp1 = 10'b0111000001;
13'b010101010zzzz : data_tmp1 = 10'b0111000001;
13'b010101011100z : data_tmp1 = 10'b0111000001;

13'b0101010111z1z : data_tmp1 = 10'b0111000010;
13'b010101011110z : data_tmp1 = 10'b0111000010;
13'b01010110zzzzz : data_tmp1 = 10'b0111000010;
13'b0101011100zzz : data_tmp1 = 10'b0111000010;
13'b010101110100z : data_tmp1 = 10'b0111000010;
13'b0101011101010 : data_tmp1 = 10'b0111000010;

13'b0101011101z11 : data_tmp1 = 10'b0111000011;
13'b01010111z110z : data_tmp1 = 10'b0111000011;
13'b01010111z1110 : data_tmp1 = 10'b0111000011;
13'b0101011110zzz : data_tmp1 = 10'b0111000011;
13'b01010111110zz : data_tmp1 = 10'b0111000011;
13'b0101011111111 : data_tmp1 = 10'b0111000011;
13'b010110000zzzz : data_tmp1 = 10'b0111000011;
13'b0101100010zzz : data_tmp1 = 10'b0111000011;
13'b01011000110zz : data_tmp1 = 10'b0111000011;

13'b0101100z111zz : data_tmp1 = 10'b0111000100;
13'b010110010zzzz : data_tmp1 = 10'b0111000100;
13'b0101100110zzz : data_tmp1 = 10'b0111000100;
13'b01011001110zz : data_tmp1 = 10'b0111000100;
13'b0101101000zzz : data_tmp1 = 10'b0111000100;
```

```
13'b01011010010zz : data_tmp1 = 10'b0111000100;
13'b010110100110z : data_tmp1 = 10'b0111000100;
13'b0101101001110 : data_tmp1 = 10'b0111000100;

13'b01011010z1111 : data_tmp1 = 10'b0111000101;
13'b0101101z10zzz : data_tmp1 = 10'b0111000101;
13'b0101101z110zz : data_tmp1 = 10'b0111000101;
13'b0101101z1110z : data_tmp1 = 10'b0111000101;
13'b0101101z11110 : data_tmp1 = 10'b0111000101;
13'b010110110zzzz : data_tmp1 = 10'b0111000101;
13'b0101101111111 : data_tmp1 = 10'b0111000101;
13'b010111000000z : data_tmp1 = 10'b0111000101;

13'b0101110000z1z : data_tmp1 = 10'b0111000110;
13'b010111000z10z : data_tmp1 = 10'b0111000110;
13'b01011100z10zz : data_tmp1 = 10'b0111000110;
13'b01011100z111z : data_tmp1 = 10'b0111000110;
13'b0101110010zzz : data_tmp1 = 10'b0111000110;
13'b010111001110z : data_tmp1 = 10'b0111000110;
13'b010111010zzzz : data_tmp1 = 10'b0111000110;
13'b01011101100zz : data_tmp1 = 10'b0111000110;
13'b010111011010z : data_tmp1 = 10'b0111000110;
13'b0101110110110 : data_tmp1 = 10'b0111000110;

13'b010111011z111 : data_tmp1 = 10'b0111000111;
13'b01011101110zz : data_tmp1 = 10'b0111000111;
13'b010111011110z : data_tmp1 = 10'b0111000111;
13'b0101110111110 : data_tmp1 = 10'b0111000111;
13'b01011110zzzzz : data_tmp1 = 10'b0111000111;
13'b0101111100zzz : data_tmp1 = 10'b0111000111;
13'b01011111010zz : data_tmp1 = 10'b0111000111;

13'b01011111z11zz : data_tmp1 = 10'b0111001000;
13'b0101111110zzz : data_tmp1 = 10'b0111001000;
13'b01011111110zz : data_tmp1 = 10'b0111001000;
13'b01100000zzzzz : data_tmp1 = 10'b0111001000;
13'b011000010000z : data_tmp1 = 10'b0111001000;

13'b0110000100z1z : data_tmp1 = 10'b0111001001;
13'b011000010z10z : data_tmp1 = 10'b0111001001;
13'b01100001z10zz : data_tmp1 = 10'b0111001001;
13'b01100001z111z : data_tmp1 = 10'b0111001001;
13'b0110000110zzz : data_tmp1 = 10'b0111001001;
13'b011000011110z : data_tmp1 = 10'b0111001001;
13'b011000100zzzz : data_tmp1 = 10'b0111001001;
13'b0110001010zzz : data_tmp1 = 10'b0111001001;
13'b0110001011000 : data_tmp1 = 10'b0111001001;

13'b01100010110z1 : data_tmp1 = 10'b0111001010;
13'b0110001011z10 : data_tmp1 = 10'b0111001010;
13'b0110001z1110z : data_tmp1 = 10'b0111001010;
13'b0110001z11111 : data_tmp1 = 10'b0111001010;
13'b011000110zzzz : data_tmp1 = 10'b0111001010;
13'b0110001110zzz : data_tmp1 = 10'b0111001010;
13'b01100011110zz : data_tmp1 = 10'b0111001010;
13'b0110001111110 : data_tmp1 = 10'b0111001010;
```

```
13'b011001000zzzz : data_tmp1 = 10'b0111001010;
13'b0110010010000 : data_tmp1 = 10'b0111001010;

13'b01100100100z1 : data_tmp1 = 10'b0111001011;
13'b0110010010z10 : data_tmp1 = 10'b0111001011;
13'b011001001z10z : data_tmp1 = 10'b0111001011;
13'b011001001z111 : data_tmp1 = 10'b0111001011;
13'b0110010z110zz : data_tmp1 = 10'b0111001011;
13'b0110010z11110 : data_tmp1 = 10'b0111001011;
13'b011001010zzzz : data_tmp1 = 10'b0111001011;
13'b0110010110zzz : data_tmp1 = 10'b0111001011;
13'b011001011110z : data_tmp1 = 10'b0111001011;
13'b0110010111111 : data_tmp1 = 10'b0111001011;
13'b0110011000zzz : data_tmp1 = 10'b0111001011;
13'b011001100100z : data_tmp1 = 10'b0111001011;
13'b0110011001010 : data_tmp1 = 10'b0111001011;

13'b0110011001z11 : data_tmp1 = 10'b0111001100;
13'b01100110z110z : data_tmp1 = 10'b0111001100;
13'b01100110z1110 : data_tmp1 = 10'b0111001100;
13'b0110011z10zzz : data_tmp1 = 10'b0111001100;
13'b0110011z110zz : data_tmp1 = 10'b0111001100;
13'b0110011z11111 : data_tmp1 = 10'b0111001100;
13'b011001110zzzz : data_tmp1 = 10'b0111001100;
13'b011001111110z : data_tmp1 = 10'b0111001100;
13'b0110011111110 : data_tmp1 = 10'b0111001100;
13'b01101000000zz : data_tmp1 = 10'b0111001100;
13'b0110100000100 : data_tmp1 = 10'b0111001100;

13'b01101000001z1 : data_tmp1 = 10'b0111001101;
13'b011010000z110 : data_tmp1 = 10'b0111001101;
13'b01101000z10zz : data_tmp1 = 10'b0111001101;
13'b01101000z110z : data_tmp1 = 10'b0111001101;
13'b01101000z1111 : data_tmp1 = 10'b0111001101;
13'b0110100z10zzz : data_tmp1 = 10'b0111001101;
13'b0110100z11110 : data_tmp1 = 10'b0111001101;
13'b011010010zzzz : data_tmp1 = 10'b0111001101;
13'b01101001110zz : data_tmp1 = 10'b0111001101;
13'b011010011110z : data_tmp1 = 10'b0111001101;
13'b0110100111111 : data_tmp1 = 10'b0111001101;

13'b01101010zzzzz : data_tmp1 = 10'b0111001110;
13'b011010110zzzz : data_tmp1 = 10'b0111001110;
13'b0110101110zzz : data_tmp1 = 10'b0111001110;
13'b01101011110zz : data_tmp1 = 10'b0111001110;

13'b01101011111zz : data_tmp1 = 10'b0111001111;
13'b01101100zzzzz : data_tmp1 = 10'b0111001111;
13'b011011010zzzz : data_tmp1 = 10'b0111001111;
13'b0110110110zzz : data_tmp1 = 10'b0111001111;
13'b0110110111000 : data_tmp1 = 10'b0111001111;

13'b01101101110z1 : data_tmp1 = 10'b0111010000;
13'b0110110111z10 : data_tmp1 = 10'b0111010000;
13'b011011011110z : data_tmp1 = 10'b0111010000;
13'b0110110111111 : data_tmp1 = 10'b0111010000;
```

```
13'b01101110zzzzz : data_tmp1 = 10'b0111010000;
13'b011011110zzzz : data_tmp1 = 10'b0111010000;
13'b0110111110zzz : data_tmp1 = 10'b0111010000;

13'b0110111111zzz : data_tmp1 = 10'b0111010001;
13'b01110000zzzzz : data_tmp1 = 10'b0111010001;
13'b011100010zzzz : data_tmp1 = 10'b0111010001;
13'b01110001100zz : data_tmp1 = 10'b0111010001;
13'b011100011010z : data_tmp1 = 10'b0111010001;
13'b0111000110110 : data_tmp1 = 10'b0111010001;

13'b011100011z111 : data_tmp1 = 10'b0111010010;
13'b01110001110zz : data_tmp1 = 10'b0111010010;
13'b011100011110z : data_tmp1 = 10'b0111010010;
13'b0111000111110 : data_tmp1 = 10'b0111010010;
13'b01110010zzzzz : data_tmp1 = 10'b0111010010;
13'b011100110zzzz : data_tmp1 = 10'b0111010010;
13'b0111001110zzz : data_tmp1 = 10'b0111010010;

13'b0111001111zzz : data_tmp1 = 10'b0111010011;
13'b01110100zzzzz : data_tmp1 = 10'b0111010011;
13'b011101010zzzz : data_tmp1 = 10'b0111010011;
13'b0111010110zzz : data_tmp1 = 10'b0111010011;
13'b011101011100z : data_tmp1 = 10'b0111010011;

13'b0111010111z1z : data_tmp1 = 10'b0111010100;
13'b011101011110z : data_tmp1 = 10'b0111010100;
13'b01110110zzzzz : data_tmp1 = 10'b0111010100;
13'b011101110zzzz : data_tmp1 = 10'b0111010100;
13'b0111011110zzz : data_tmp1 = 10'b0111010100;
13'b01110111110zz : data_tmp1 = 10'b0111010100;
13'b0111011111100 : data_tmp1 = 10'b0111010100;

13'b01110111111z1 : data_tmp1 = 10'b0111010101;
13'b0111011111110 : data_tmp1 = 10'b0111010101;
13'b0111100zzzzzz : data_tmp1 = 10'b0111010101;
13'b0111101000000 : data_tmp1 = 10'b0111010101;

13'b01111010000z1 : data_tmp1 = 10'b0111010110;
13'b0111101000z10 : data_tmp1 = 10'b0111010110;
13'b011110100z10z : data_tmp1 = 10'b0111010110;
13'b011110100z111 : data_tmp1 = 10'b0111010110;
13'b01111010z10zz : data_tmp1 = 10'b0111010110;
13'b01111010z1110 : data_tmp1 = 10'b0111010110;
13'b0111101z10zzz : data_tmp1 = 10'b0111010110;
13'b0111101z1110z : data_tmp1 = 10'b0111010110;
13'b0111101z11111 : data_tmp1 = 10'b0111010110;
13'b011110110zzzz : data_tmp1 = 10'b0111010110;
13'b01111011110zz : data_tmp1 = 10'b0111010110;
13'b0111101111110 : data_tmp1 = 10'b0111010110;
13'b01111100000zz : data_tmp1 = 10'b0111010110;
13'b011111000010z : data_tmp1 = 10'b0111010110;

13'b011111000z11z : data_tmp1 = 10'b0111010111;
13'b01111100z10zz : data_tmp1 = 10'b0111010111;
13'b01111100z110z : data_tmp1 = 10'b0111010111;
```

```
13'b0111110z10zzz : data_tmp1 = 10'b0111010111;
13'b0111110z1111z : data_tmp1 = 10'b0111010111;
13'b011111010zzzz : data_tmp1 = 10'b0111010111;
13'b01111101110zz : data_tmp1 = 10'b0111010111;
13'b011111011110z : data_tmp1 = 10'b0111010111;
13'b0111111000zzz : data_tmp1 = 10'b0111010111;
13'b01111110010zz : data_tmp1 = 10'b0111010111;
13'b0111111001100 : data_tmp1 = 10'b0111010111;

13'b01111110011z1 : data_tmp1 = 10'b0111011000;
13'b01111110z1110 : data_tmp1 = 10'b0111011000;
13'b0111111z10zzz : data_tmp1 = 10'b0111011000;
13'b0111111z110zz : data_tmp1 = 10'b0111011000;
13'b0111111z1110z : data_tmp1 = 10'b0111011000;
13'b0111111z11111 : data_tmp1 = 10'b0111011000;
13'b011111110zzzz : data_tmp1 = 10'b0111011000;
13'b0111111111110 : data_tmp1 = 10'b0111011000;
13'b100000000zzzz : data_tmp1 = 10'b0111011000;
13'b1000000100zz : data_tmp1 = 10'b0111011000;

13'b100000001z1zz : data_tmp1 = 10'b0111011001;
13'b1000000z110zz : data_tmp1 = 10'b0111011001;
13'b100000010zzzz : data_tmp1 = 10'b0111011001;
13'b1000000110zzz : data_tmp1 = 10'b0111011001;
13'b10000001111zz : data_tmp1 = 10'b0111011001;
13'b100000100zzzz : data_tmp1 = 10'b0111011001;
13'b1000001010zzz : data_tmp1 = 10'b0111011001;
13'b10000010110zz : data_tmp1 = 10'b0111011001;
13'b100000101110z : data_tmp1 = 10'b0111011001;

13'b1000001z1111z : data_tmp1 = 10'b0111011010;
13'b100000110zzzz : data_tmp1 = 10'b0111011010;
13'b1000001110zzz : data_tmp1 = 10'b0111011010;
13'b10000011110zz : data_tmp1 = 10'b0111011010;
13'b100000111110z : data_tmp1 = 10'b0111011010;
13'b10000100zzzzz : data_tmp1 = 10'b0111011010;
13'b1000010100zzz : data_tmp1 = 10'b0111011010;

13'b10000101z1zzz : data_tmp1 = 10'b0111011011;
13'b1000010110zzz : data_tmp1 = 10'b0111011011;
13'b10000110zzzzz : data_tmp1 = 10'b0111011011;
13'b100001110zzzz : data_tmp1 = 10'b0111011011;
13'b10000111100zz : data_tmp1 = 10'b0111011011;

13'b100001111z1zz : data_tmp1 = 10'b0111011100;
13'b10000111110zz : data_tmp1 = 10'b0111011100;
13'b1000100zzzzzz : data_tmp1 = 10'b0111011100;
13'b1000101000000 : data_tmp1 = 10'b0111011100;

13'b10001010000z1 : data_tmp1 = 10'b0111011101;
13'b1000101000z10 : data_tmp1 = 10'b0111011101;
13'b100010100z10z : data_tmp1 = 10'b0111011101;
13'b100010100z111 : data_tmp1 = 10'b0111011101;
13'b10001010z10zz : data_tmp1 = 10'b0111011101;
13'b10001010z1110 : data_tmp1 = 10'b0111011101;
13'b1000101z10zzz : data_tmp1 = 10'b0111011101;
```

```
13'b1000101z1110z : data_tmp1 = 10'b0111011101;
13'b1000101z11111 : data_tmp1 = 10'b0111011101;
13'b100010110zzzz : data_tmp1 = 10'b0111011101;
13'b1000101110zz : data_tmp1 = 10'b0111011101;
13'b1000101111110 : data_tmp1 = 10'b0111011101;
13'b100011000zzz : data_tmp1 = 10'b0111011101;
13'b10001100010zz : data_tmp1 = 10'b0111011101;
13'b100011000110z : data_tmp1 = 10'b0111011101;
13'b1000110001110 : data_tmp1 = 10'b0111011101;

13'b10001100z1111 : data_tmp1 = 10'b0111011110;
13'b1000110z10zzz : data_tmp1 = 10'b0111011110;
13'b1000110z110zz : data_tmp1 = 10'b0111011110;
13'b1000110z1110z : data_tmp1 = 10'b0111011110;
13'b1000110z11110 : data_tmp1 = 10'b0111011110;
13'b100011010zzzz : data_tmp1 = 10'b0111011110;
13'b1000110111111 : data_tmp1 = 10'b0111011110;
13'b100011100zzzz : data_tmp1 = 10'b0111011110;
13'b1000111010zzz : data_tmp1 = 10'b0111011110;
13'b10001110110zz : data_tmp1 = 10'b0111011110;
13'b100011101110z : data_tmp1 = 10'b0111011110;
13'b1000111011110 : data_tmp1 = 10'b0111011110;

13'b1000111z11111 : data_tmp1 = 10'b0111011111;
13'b100011110zzzz : data_tmp1 = 10'b0111011111;
13'b1000111110zzz : data_tmp1 = 10'b0111011111;
13'b10001111110zz : data_tmp1 = 10'b0111011111;
13'b100011111110z : data_tmp1 = 10'b0111011111;
13'b1000111111110 : data_tmp1 = 10'b0111011111;
13'b10010000zzzzz : data_tmp1 = 10'b0111011111;
13'b100100010zzzz : data_tmp1 = 10'b0111011111;

13'b100100z11zzzz : data_tmp1 = 10'b0111100000;
13'b10010010zzzzz : data_tmp1 = 10'b0111100000;
13'b100100110zzzz : data_tmp1 = 10'b0111100000;
13'b100101000000z : data_tmp1 = 10'b0111100000;
13'b1001010000010 : data_tmp1 = 10'b0111100000;

13'b1001010000z11 : data_tmp1 = 10'b0111100001;
13'b100101000z10z : data_tmp1 = 10'b0111100001;
13'b100101000z110 : data_tmp1 = 10'b0111100001;
13'b10010100z10zz : data_tmp1 = 10'b0111100001;
13'b10010100z1111 : data_tmp1 = 10'b0111100001;
13'b1001010z10zzz : data_tmp1 = 10'b0111100001;
13'b1001010z1110z : data_tmp1 = 10'b0111100001;
13'b1001010z11110 : data_tmp1 = 10'b0111100001;
13'b100101010zzzz : data_tmp1 = 10'b0111100001;
13'b10010101110zz : data_tmp1 = 10'b0111100001;
13'b1001010111111 : data_tmp1 = 10'b0111100001;
13'b100101100zzzz : data_tmp1 = 10'b0111100001;
13'b10010110100zz : data_tmp1 = 10'b0111100001;
13'b100101101010z : data_tmp1 = 10'b0111100001;
13'b1001011010110 : data_tmp1 = 10'b0111100001;

13'b100101101z111 : data_tmp1 = 10'b0111100010;
13'b1001011z110zz : data_tmp1 = 10'b0111100010;
```

```
13'b1001011z1110z : data_tmp1 = 10'b0111100010;
13'b1001011z11110 : data_tmp1 = 10'b0111100010;
13'b100101110zzzz : data_tmp1 = 10'b0111100010;
13'b1001011110zzz : data_tmp1 = 10'b0111100010;
13'b1001011111111 : data_tmp1 = 10'b0111100010;
13'b10011000zzzzz : data_tmp1 = 10'b0111100010;
13'b1001100100zzz : data_tmp1 = 10'b0111100010;
13'b10011001010zz : data_tmp1 = 10'b0111100010;
13'b1001100101100 : data_tmp1 = 10'b0111100010;

13'b10011001011z1 : data_tmp1 = 10'b0111100011;
13'b10011001z1110 : data_tmp1 = 10'b0111100011;
13'b100110z110zzz : data_tmp1 = 10'b0111100011;
13'b100110z1110zz : data_tmp1 = 10'b0111100011;
13'b100110z11110z : data_tmp1 = 10'b0111100011;
13'b100110z111111 : data_tmp1 = 10'b0111100011;
13'b10011010zzzzz : data_tmp1 = 10'b0111100011;
13'b100110110zzzz : data_tmp1 = 10'b0111100011;
13'b1001101111110 : data_tmp1 = 10'b0111100011;
13'b10011100000zz : data_tmp1 = 10'b0111100011;

13'b100111000z1zz : data_tmp1 = 10'b0111100100;
13'b10011100z10zz : data_tmp1 = 10'b0111100100;
13'b1001110z10zzz : data_tmp1 = 10'b0111100100;
13'b1001110z111zz : data_tmp1 = 10'b0111100100;
13'b100111010zzzz : data_tmp1 = 10'b0111100100;
13'b10011101110zz : data_tmp1 = 10'b0111100100;
13'b100111100zzzz : data_tmp1 = 10'b0111100100;
13'b1001111010zzz : data_tmp1 = 10'b0111100100;
13'b10011110110zz : data_tmp1 = 10'b0111100100;
13'b1001111011100 : data_tmp1 = 10'b0111100100;

13'b10011110111z1 : data_tmp1 = 10'b0111100101;
13'b1001111z11110 : data_tmp1 = 10'b0111100101;
13'b100111110zzzz : data_tmp1 = 10'b0111100101;
13'b1001111110zzz : data_tmp1 = 10'b0111100101;
13'b10011111110zz : data_tmp1 = 10'b0111100101;
13'b100111111110z : data_tmp1 = 10'b0111100101;
13'b1001111111111 : data_tmp1 = 10'b0111100101;
13'b10100000zzzzz : data_tmp1 = 10'b0111100101;
13'b101000010zzzz : data_tmp1 = 10'b0111100101;
13'b10100001100zz : data_tmp1 = 10'b0111100101;
13'b101000011010z : data_tmp1 = 10'b0111100101;
13'b1010000110110 : data_tmp1 = 10'b0111100101;

13'b101000011z111 : data_tmp1 = 10'b0111100110;
13'b101000z1110zz : data_tmp1 = 10'b0111100110;
13'b101000z11110z : data_tmp1 = 10'b0111100110;
13'b101000z111110 : data_tmp1 = 10'b0111100110;
13'b10100010zzzzz : data_tmp1 = 10'b0111100110;
13'b101000110zzzz : data_tmp1 = 10'b0111100110;
13'b1010001110zzz : data_tmp1 = 10'b0111100110;
13'b1010001111111 : data_tmp1 = 10'b0111100110;
13'b101001000zzzz : data_tmp1 = 10'b0111100110;
13'b101001001000z : data_tmp1 = 10'b0111100110;
13'b1010010010010 : data_tmp1 = 10'b0111100110;
```

```
13'b1010010010z11 : data_tmp1 = 10'b0111100111;
13'b101001001z10z : data_tmp1 = 10'b0111100111;
13'b101001001z110 : data_tmp1 = 10'b0111100111;
13'b1010010z110zz : data_tmp1 = 10'b0111100111;
13'b1010010z11111 : data_tmp1 = 10'b0111100111;
13'b101001z10zzzz : data_tmp1 = 10'b0111100111;
13'b1010010110zzz : data_tmp1 = 10'b0111100111;
13'b101001011110z : data_tmp1 = 10'b0111100111;
13'b1010010111110 : data_tmp1 = 10'b0111100111;
13'b10100110zzzzz : data_tmp1 = 10'b0111100111;

13'b101001111zzzz : data_tmp1 = 10'b0111101000;
13'b1010100zzzzzz : data_tmp1 = 10'b0111101000;
13'b101010100zzzz : data_tmp1 = 10'b0111101000;

13'b1010101z1zzzz : data_tmp1 = 10'b0111101001;
13'b101010110zzzz : data_tmp1 = 10'b0111101001;
13'b10101100zzzzz : data_tmp1 = 10'b0111101001;
13'b101011010zzzz : data_tmp1 = 10'b0111101001;

13'b101011z11zzzz : data_tmp1 = 10'b0111101010;
13'b10101110zzzzz : data_tmp1 = 10'b0111101010;
13'b101011110zzzz : data_tmp1 = 10'b0111101010;
13'b101100000zzzz : data_tmp1 = 10'b0111101010;
13'b101100001000z : data_tmp1 = 10'b0111101010;
13'b1011000010010 : data_tmp1 = 10'b0111101010;

13'b1011000010z11 : data_tmp1 = 10'b0111101011;
13'b101100001z10z : data_tmp1 = 10'b0111101011;
13'b101100001z110 : data_tmp1 = 10'b0111101011;
13'b1011000z110zz : data_tmp1 = 10'b0111101011;
13'b1011000z11111 : data_tmp1 = 10'b0111101011;
13'b101100z10zzzz : data_tmp1 = 10'b0111101011;
13'b1011000110zzz : data_tmp1 = 10'b0111101011;
13'b101100011110z : data_tmp1 = 10'b0111101011;
13'b1011000111110 : data_tmp1 = 10'b0111101011;
13'b10110010zzzzz : data_tmp1 = 10'b0111101011;
13'b10110011100zz : data_tmp1 = 10'b0111101011;
13'b101100111010z : data_tmp1 = 10'b0111101011;
13'b1011001110110 : data_tmp1 = 10'b0111101011;

13'b101100111z111 : data_tmp1 = 10'b0111101100;
13'b10110011110zz : data_tmp1 = 10'b0111101100;
13'b101100111110z : data_tmp1 = 10'b0111101100;
13'b1011001111110 : data_tmp1 = 10'b0111101100;
13'b1011010zzzzzz : data_tmp1 = 10'b0111101100;
13'b101101100zzzz : data_tmp1 = 10'b0111101100;
13'b1011011010zzz : data_tmp1 = 10'b0111101100;
13'b10110110110zz : data_tmp1 = 10'b0111101100;
13'b101101101110z : data_tmp1 = 10'b0111101100;

13'b1011011z1111z : data_tmp1 = 10'b0111101101;
13'b101101110zzzz : data_tmp1 = 10'b0111101101;
13'b1011011110zzz : data_tmp1 = 10'b0111101101;
13'b10110111110zz : data_tmp1 = 10'b0111101101;
13'b101101111110z : data_tmp1 = 10'b0111101101;
```

```
13'b1011100zzzzzz : data_tmp1 = 10'b0111101101;
13'b10111010000zz : data_tmp1 = 10'b0111101101;
13'b101110100010z : data_tmp1 = 10'b0111101101;

13'b101110100z11z : data_tmp1 = 10'b0111101110;
13'b10111010z10zz : data_tmp1 = 10'b0111101110;
13'b10111010z110z : data_tmp1 = 10'b0111101110;
13'b1011101z10zzz : data_tmp1 = 10'b0111101110;
13'b1011101z1111z : data_tmp1 = 10'b0111101110;
13'b101110110zzzz : data_tmp1 = 10'b0111101110;
13'b10111011110zz : data_tmp1 = 10'b0111101110;
13'b101110111110z : data_tmp1 = 10'b0111101110;
13'b10111100zzzzz : data_tmp1 = 10'b0111101110;
13'b1011110100zzz : data_tmp1 = 10'b0111101110;
13'b10111101010zz : data_tmp1 = 10'b0111101110;
13'b101111010110z : data_tmp1 = 10'b0111101110;
13'b1011110101110 : data_tmp1 = 10'b0111101110;

13'b10111101z1111 : data_tmp1 = 10'b0111101111;
13'b101111z110zzz : data_tmp1 = 10'b0111101111;
13'b101111z1110zz : data_tmp1 = 10'b0111101111;
13'b101111z11110z : data_tmp1 = 10'b0111101111;
13'b101111z111110 : data_tmp1 = 10'b0111101111;
13'b10111110zzzzz : data_tmp1 = 10'b0111101111;
13'b101111110zzzz : data_tmp1 = 10'b0111101111;
13'b1011111111111 : data_tmp1 = 10'b0111101111;
13'b110000000zzzz : data_tmp1 = 10'b0111101111;
13'b1100000010zzz : data_tmp1 = 10'b0111101111;
13'b110000001100z : data_tmp1 = 10'b0111101111;
13'b1100000011010 : data_tmp1 = 10'b0111101111;

13'b1100000011z11 : data_tmp1 = 10'b0111110000;
13'b1100000z1110z : data_tmp1 = 10'b0111110000;
13'b1100000z11110 : data_tmp1 = 10'b0111110000;
13'b110000z10zzzz : data_tmp1 = 10'b0111110000;
13'b110000z110zzz : data_tmp1 = 10'b0111110000;
13'b110000z1110zz : data_tmp1 = 10'b0111110000;
13'b110000z111111 : data_tmp1 = 10'b0111110000;
13'b11000010zzzzz : data_tmp1 = 10'b0111110000;
13'b110000111110z : data_tmp1 = 10'b0111110000;
13'b1100001111110 : data_tmp1 = 10'b0111110000;
13'b1100010000zzz : data_tmp1 = 10'b0111110000;
13'b1100010001000 : data_tmp1 = 10'b0111110000;

13'b11000100010z1 : data_tmp1 = 10'b0111110001;
13'b1100010001z10 : data_tmp1 = 10'b0111110001;
13'b11000100z110z : data_tmp1 = 10'b0111110001;
13'b11000100z1111 : data_tmp1 = 10'b0111110001;
13'b1100010z10zzz : data_tmp1 = 10'b0111110001;
13'b1100010z110zz : data_tmp1 = 10'b0111110001;
13'b1100010z11110 : data_tmp1 = 10'b0111110001;
13'b110001z10zzzz : data_tmp1 = 10'b0111110001;
13'b110001011110z : data_tmp1 = 10'b0111110001;
13'b1100010111111 : data_tmp1 = 10'b0111110001;
13'b11000110zzzzz : data_tmp1 = 10'b0111110001;
13'b1100011110zzz : data_tmp1 = 10'b0111110001;
```

```
13'b1100011111zzz : data_tmp1 = 10'b0111110010;
13'b1100100zzzzzz : data_tmp1 = 10'b0111110010;
13'b11001010zzzzz : data_tmp1 = 10'b0111110010;
13'b1100101100zzz : data_tmp1 = 10'b0111110010;
13'b110010110100z : data_tmp1 = 10'b0111110010;

13'b1100101101z1z : data_tmp1 = 10'b0111110011;
13'b11001011z110z : data_tmp1 = 10'b0111110011;
13'b1100101110zzz : data_tmp1 = 10'b0111110011;
13'b11001011110zz : data_tmp1 = 10'b0111110011;
13'b110010111111z : data_tmp1 = 10'b0111110011;
13'b1100110zzzzzz : data_tmp1 = 10'b0111110011;
13'b110011100zzzz : data_tmp1 = 10'b0111110011;
13'b1100111010zzz : data_tmp1 = 10'b0111110011;
13'b11001110110zz : data_tmp1 = 10'b0111110011;
13'b1100111011100 : data_tmp1 = 10'b0111110011;

13'b11001110111z1 : data_tmp1 = 10'b0111110100;
13'b1100111z11110 : data_tmp1 = 10'b0111110100;
13'b110011110zzzz : data_tmp1 = 10'b0111110100;
13'b1100111110zzz : data_tmp1 = 10'b0111110100;
13'b11001111110zz : data_tmp1 = 10'b0111110100;
13'b110011111110z : data_tmp1 = 10'b0111110100;
13'b1100111111111 : data_tmp1 = 10'b0111110100;
13'b1101000zzzzzz : data_tmp1 = 10'b0111110100;
13'b110100100zzzz : data_tmp1 = 10'b0111110100;
13'b110100101000z : data_tmp1 = 10'b0111110100;
13'b1101001010010 : data_tmp1 = 10'b0111110100;

13'b1101001010z11 : data_tmp1 = 10'b0111110101;
13'b110100101z10z : data_tmp1 = 10'b0111110101;
13'b110100101z110 : data_tmp1 = 10'b0111110101;
13'b1101001z110zz : data_tmp1 = 10'b0111110101;
13'b1101001z11111 : data_tmp1 = 10'b0111110101;
13'b110100110zzzz : data_tmp1 = 10'b0111110101;
13'b1101001110zzz : data_tmp1 = 10'b0111110101;
13'b110100111110z : data_tmp1 = 10'b0111110101;
13'b1101001111110 : data_tmp1 = 10'b0111110101;
13'b1101010zzzzzz : data_tmp1 = 10'b0111110101;
13'b1101011000zzz : data_tmp1 = 10'b0111110101;
13'b110101100100z : data_tmp1 = 10'b0111110101;

13'b1101011001z1z : data_tmp1 = 10'b0111110110;
13'b11010110z110z : data_tmp1 = 10'b0111110110;
13'b1101011z10zzz : data_tmp1 = 10'b0111110110;
13'b1101011z110zz : data_tmp1 = 10'b0111110110;
13'b1101011z1111z : data_tmp1 = 10'b0111110110;
13'b110101110zzzz : data_tmp1 = 10'b0111110110;
13'b110101111110z : data_tmp1 = 10'b0111110110;
13'b1101100zzzzzz : data_tmp1 = 10'b0111110110;
13'b11011010000zz : data_tmp1 = 10'b0111110110;

13'b110110100z1zz : data_tmp1 = 10'b0111110111;
13'b11011010z10zz : data_tmp1 = 10'b0111110111;
13'b1101101z10zzz : data_tmp1 = 10'b0111110111;
13'b1101101z111zz : data_tmp1 = 10'b0111110111;
```

```
13'b110110110zzzz : data_tmp1 = 10'b0111110111;
13'b11011011110zz : data_tmp1 = 10'b0111110111;
13'b1101110zzzzzz : data_tmp1 = 10'b0111110111;

13'b1101111zzzzzz : data_tmp1 = 10'b0111111000;
13'b11100000zzzzz : data_tmp1 = 10'b0111111000;
13'b111000010zzzz : data_tmp1 = 10'b0111111000;
13'b1110000110zzz : data_tmp1 = 10'b0111111000;
13'b11100001110zz : data_tmp1 = 10'b0111111000;
13'b111000011110z : data_tmp1 = 10'b0111111000;

13'b111000z11111z : data_tmp1 = 10'b0111111001;
13'b11100010zzzzz : data_tmp1 = 10'b0111111001;
13'b111000110zzzz : data_tmp1 = 10'b0111111001;
13'b1110001110zzz : data_tmp1 = 10'b0111111001;
13'b11100011110zz : data_tmp1 = 10'b0111111001;
13'b111000111110z : data_tmp1 = 10'b0111111001;
13'b11100100zzzzz : data_tmp1 = 10'b0111111001;
13'b111001010zzzz : data_tmp1 = 10'b0111111001;
13'b1110010110zzz : data_tmp1 = 10'b0111111001;
13'b11100101110zz : data_tmp1 = 10'b0111111001;
13'b111001011110z : data_tmp1 = 10'b0111111001;
13'b1110010111110 : data_tmp1 = 10'b0111111001;

13'b111001z111111 : data_tmp1 = 10'b0111111010;
13'b11100110zzzzz : data_tmp1 = 10'b0111111010;
13'b111001110zzzz : data_tmp1 = 10'b0111111010;
13'b1110011110zzz : data_tmp1 = 10'b0111111010;
13'b11100111110zz : data_tmp1 = 10'b0111111010;
13'b111001111110z : data_tmp1 = 10'b0111111010;
13'b1110011111110 : data_tmp1 = 10'b0111111010;
13'b1110100zzzzzz : data_tmp1 = 10'b0111111010;
13'b111010100000z : data_tmp1 = 10'b0111111010;

13'b1110101000z1z : data_tmp1 = 10'b0111111011;
13'b111010100z10z : data_tmp1 = 10'b0111111011;
13'b11101010z10zz : data_tmp1 = 10'b0111111011;
13'b11101010z111z : data_tmp1 = 10'b0111111011;
13'b1110101z10zzz : data_tmp1 = 10'b0111111011;
13'b1110101z1110z : data_tmp1 = 10'b0111111011;
13'b111010110zzzz : data_tmp1 = 10'b0111111011;
13'b11101011110zz : data_tmp1 = 10'b0111111011;
13'b111010111111z : data_tmp1 = 10'b0111111011;
13'b1110110zzzzzz : data_tmp1 = 10'b0111111011;
13'b11101110000zz : data_tmp1 = 10'b0111111011;
13'b111011100010z : data_tmp1 = 10'b0111111011;
13'b1110111000110 : data_tmp1 = 10'b0111111011;

13'b111011100z111 : data_tmp1 = 10'b0111111100;
13'b11101110z10zz : data_tmp1 = 10'b0111111100;
13'b11101110z110z : data_tmp1 = 10'b0111111100;
13'b11101110z1110 : data_tmp1 = 10'b0111111100;
13'b1110111z10zzz : data_tmp1 = 10'b0111111100;
13'b1110111z11111 : data_tmp1 = 10'b0111111100;
13'b111011110zzzz : data_tmp1 = 10'b0111111100;
13'b11101111110zz : data_tmp1 = 10'b0111111100;
```

```
13'b111011111110z : data_tmp1 = 10'b0111111100;
13'b1110111111110 : data_tmp1 = 10'b0111111100;
13'b1111000zzzzzz : data_tmp1 = 10'b0111111100;
13'b1111001000zzz : data_tmp1 = 10'b0111111100;
13'b11110010010zz : data_tmp1 = 10'b0111111100;
13'b111100100110z : data_tmp1 = 10'b0111111100;

13'b11110010z111z : data_tmp1 = 10'b0111111101;
13'b1111001z10zzz : data_tmp1 = 10'b0111111101;
13'b1111001z110zz : data_tmp1 = 10'b0111111101;
13'b1111001z1110z : data_tmp1 = 10'b0111111101;
13'b111100110zzzz : data_tmp1 = 10'b0111111101;
13'b111100111111z : data_tmp1 = 10'b0111111101;
13'b1111010zzzzzz : data_tmp1 = 10'b0111111101;
13'b111101100zzzz : data_tmp1 = 10'b0111111101;
13'b1111011010zzz : data_tmp1 = 10'b0111111101;

13'b1111011z11zzz : data_tmp1 = 10'b0111111110;
13'b111101110zzzz : data_tmp1 = 10'b0111111110;
13'b1111011110zzz : data_tmp1 = 10'b0111111110;
13'b1111100zzzzzz : data_tmp1 = 10'b0111111110;
13'b11111010zzzzz : data_tmp1 = 10'b0111111110;
13'b11111011000zz : data_tmp1 = 10'b0111111110;

13'b111110110z1zz : data_tmp1 = 10'b0111111111;
13'b11111011z10zz : data_tmp1 = 10'b0111111111;
13'b11111z1110zzz : data_tmp1 = 10'b0111111111;
13'b11111z11111zz : data_tmp1 = 10'b0111111111;
13'b111110zzzzzz : data_tmp1 = 10'b0111111111;
13'b11111110zzzzz : data_tmp1 = 10'b0111111111;
13'b111111110zzzz : data_tmp1 = 10'b0111111111;
13'b11111111110zz : data_tmp1 = 10'b0111111111;
default: data_tmp1 = 10'bxxxxxxxxxx;
 endcase

 always @(posedge clk)
  if (enable_3)
   data_tmp2 <= data_tmp1;

 assign out_data = data_tmp2;

endmodule
```

Listing 14

```
// SccsId: %W% %G%
/*******************************************************************************
   Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author  : Dawood Alam.

 Description: Verilog code for windowing algorithm to enable detection of the
       "active interval" of the COFDM symbol for guard values of:
         64, 128, 256, 512 and an active interval of 2048. (RTL)

 Notes  : This module generates the window signal for the FFT in the form
```

```
    of valid_in and provides the necessary signals for the I/Q
      demodulator, sync interpolator and error handler.

    To DO: Check between successive symbol acquires for consistency
        in timing.
        Window timing pulse
        tracking mode, filter peaks
        IQ and sync interpolator guard pulses.
        Override functions for timing.
        Gain confidence by comparing symbol_acq vs retrys

*********************************************************************************/

`timescale 1ns / 100ps

module fft_window (in_xr,
        in_xi,
            clk,
            nrst,
            valid_in,
            valid_out,
            in_resync,
        out_iqgi,
            out_sincgi,
            out_rx_guard,
            out_acquired,
            out_fft_window,
            enable_3_4,
            out_test,
            track_ram_address,
            xri_tmp1,
            xri_tmp5,
            track_ram_rnotw,
            track_ram_enable,
            ram_addr,
            ram_enable,
            ram_rnotw,
            ram10_in,
            ram10_out,
            x1r_10,          // To FFT datapath (I).
            x1i_10,          // To FFT datapath (Q).
            z2r_10,          // From FFT datapath (I)
            z2i_10,          // From FFT datapath (Q)
            fft_ram_rnotw,         // From FFT addr gen.
            fft_ram_enable,         // From FFT addr gen.
            fft_ram_addr);         // From FFT addr gen.

// ------------------------------------------------------------------
//          Parameter definitions.
// ------------------------------------------------------------------

parameter      wordlength = 12;     // Data wordlength.
parameter      r_wordlength = 10;   // ROM data wordlength.
parameter      AddressSize = 13;    // Size of address bus.
parameter      FIFO_L = 256;        // Tracking FIFO length.
parameter      FIFO_L_bits = 8;     // Track FIFO addr bits
```

```
parameter       FIFO_N = 64;       // Acc length S(i-j).
parameter       FIFO_n = 64;       // Acc length S(i-n-j).
parameter       FIFO_A = 32;       // t_offset dly FIFO+1.
parameter       FIFO_A_bits = 5;   // Track FIFO bits.
parameter       lu_AddressSize = 15;  // log lu address size.
parameter       delta = 20;        // Gu threshold distance
parameter       acquired_symbols = 2;  // Acq.symbls before trk
parameter       pos_threshold = 3;   // For info only.
parameter       t_offset_threshold = 10; // t_offset valid thresh
parameter       w_advance = 10;      // win trig frm boundary
parameter       sincint_latency = 2;   // Latency to sinc intep
parameter       iqdemod_latency = 168;  // Latency to IQ demod.

parameter        start   = 3'b000,    // Search for neg peak.
            peak1   = 3'b001,    // 1st pos peak found.
            peak2   = 3'b010,    // 2nd pos peak found.
            peak3   = 3'b011,    // 3rd pos peak found.
            track1  = 3'b100,    // Tracking mode1.
            track2  = 3'b101;    // Tracking mode1.

// ---------------------------------------------------------------------
//          Input/Output ports.
// ---------------------------------------------------------------------

input         clk,          // Master clock.
          nrst,         // Power-up reset.
            valid_in,       // Input data valid.
            in_resync,      // Sync FSM into Acqure.
            fft_ram_rnotw,
            fft_ram_enable;

input [AddressSize-1:0]  fft_ram_addr;

input [wordlength-3:0]  in_xr,         // FFT input data, I.
            in_xi,         // FFT input data, Q.
             xri_tmp5;      // Track RAM output.

input [wordlength*2-1:0] ram10_out;       // From 1K x 24 bit RAM.

input [wordlength-1:0]  z2r_10, z2i_10;     // From FFT datapath.

output [wordlength*2-1:0] ram10_in;        // To 1K x 24 bit RAM.

output [wordlength-3:0]  xri_tmp1;        // Track RAM input.

output [14:0]      out_test;          // Temp testpin output.

output          out_iqgi,       // I/Q demod guard info.
          out_sincgi,     // Sinc int. guard info.
          out_acquired,     // Symbol acquired flag.
          out_fft_window,     // FFT processor st/stp
          enable_3_4,
            valid_out,
            track_ram_rnotw,
            track_ram_enable,
            ram_enable,
```

```
ram_rnotw;

output [FIFO_L_bits-1:0] track_ram_address;   // Tracking ram address

output [1:0]      out_rx_guard;      // Acquired gu length.

output [AddressSize-1:0] ram_addr;

output [wordlength-1:0] x1r_10, x1i_10;    // To FFT datapath.

// ----------------------------------------------------------------
//        Wire/register declarations.
// ----------------------------------------------------------------

reg         out_acquired,      // Symbol acquired flag.
          out_fft_window,      // FFT window signal.
            tracking,        // Tracking mode data.
            acc_add,         // Acc add only flag.
            acc_add_sub,       // Acc add/sub flag.
            fifo_a_add_sub,      // FIFO_A add/sub flag.
          f_ratio_valid,     // F ratio is valid
            read,          // Track FIFO read flag.
            write,         // Track FIFO write flag
            track_mode,        // Track/Acq status flag
            dpctl_reset,       // Datapath control rst.
            t_reset,         // Timing counter reset.
            g_a_reset,         // Guard_active cnt rst.
            guard_valid,       // Guard signal is valid
          t_retime_acq,        // Retime timing counter
            t_retime_trk,      // Retiming for tracking
            t_offset_valid,      // Peak offset valid.
            t_offset_avg_valid,    // Average offset valid.
            pulse,         // Pulse on states 4 & 5
            enable_fft,        // FFT enabled flag.
            out_sincgi,        // Guard int to sincint.
            out_iqgi,        // Guard int to iq demod
            ram_enable,
            ram_rnotw;

reg [14:0]      guard_active;      // Guard+active length.
reg [3:0]      retry,         // No failed retry's.
          acq_symbols;       // No of acquired symbls
reg [wordlength-2:0] xri_tmp7;         // Delayed difference.
reg [wordlength-3:0] xr_reg,         // (10 bits)
          xi_reg,
            xri_tmp1,        // Sum of |I| + |Q|.
            xri_tmp3,        // Delayed |difference|.
            xri_tmp6;        // FIFO 2K/L output.

reg [FIFO_L_bits-1:0] read_address,      // Track FIFO read addr.
          write_address,     // Track FIFO write adr.
          track_ram_address;   // Tracking ram address;

reg [lu_AddressSize-1:0] acc;          // Holds input variance.
reg [wordlength-4:0] xr_tmp1,        // |I|.
          xi_tmp1;       // |Q|.
```

```
reg [2:0]        r;          // Clock decode counter.
reg [1:0]        out_rx_guard;       // Determined guard.
reg [r_wordlength:0] f_ratio;        // Statistical F ratio.
reg [10:0]       fft_valid_count;    // Counts no of FFT vlds
reg [AddressSize-1:0] window_ram_addr,   // ram_address counter.
        ram_addr;
reg [14:0]       t_count,        // Window timing count.
        t_offset;        // Peak offset from t_ct
reg [14:0]       g_a_count;      // Guard_active counter.
reg [14:0]       dp_count;       // Datapath timing count
reg [14:0]       t_offset_avg;       // Averaged offset.
reg [2:0]        state,      // Acq/Track FSM state.
        old_state;       // Old tracking state.
reg [9:0]        guard_length;       // Thresholded guard len

reg [FIFO_A_bits:0] fifo_a_count;      // Count till fifo_a ful
                // 1 bit more -> retime

reg [r_wordlength-1:0] max_peak;        // Maximum positive peak

reg [wordlength-1:0] msb_out_tmp,      // Temporary stores for
        lsb_in_tmp;      // even symbols to RAM.
wire [AddressSize-1:0] fft_ram_addr;      // From FFT RAM addr gen

wire        clk,        // Master clock.
        nrst,       // Power-up reset.
        enable_0_4,     // Clock enable 0 in 4.
        enable_1_4,     // Clock enable 1 in 4.
        enable_2_4,     // Clock enable 2 in 4.
        enable_3_4,     // Clock enable 3 in 4.
        enable_0_8,     // Clock enable 0 in 8.
        enable_1_8,     // Clock enable 1 in 8.
        enable_2_8,     // Clock enable 2 in 8.
        enable_3_8,     // Clock enable 3 in 8.
        enable_4_8,     // Clock enable 4 in 8.
        enable_5_8,     // Clock enable 5 in 8.
        enable_6_8,     // Clock enable 6 in 8.
        enable_7_8,     // Clock enable 7 in 8.
        ram_enable_8,       // Acq FIFO enable.
        track_ram_enable,   // Tracking RAM enable
        track_ram_rnotw,    // Tracking RAM rnotw.
        even_symbol,        // valid on even symbols
        in_resync,      // Resync to acqn mode.
        pos_peak,       // +ve peak, ref only!
        dp_control,     // Datapath acq/trk ctl.
        t_offset_ctl,       // Trk averager dp ctl.
        fft_ram_rnotw,
        fft_ram_enable;
wire [lu_AddressSize-1:0] lu_address;
wire [r_wordlength-1:0] lu_data,
        xri_tmp9;
wire [wordlength-3:0] xri_tmp2,
        xri_tmp4,
        xri_tmp5,
        in_q,
        out_q;
```

```
wire [wordlength-1:0] ram_in;

reg [wordlength-1:0] lsb_out,
        msb_out;

reg [wordlength-1:0] ram_out,
      msb_in,
        lsb_in;

wire [wordlength*2-1:0] ram10_out;
reg [wordlength*2-1:0] ram10_in;

reg [wordlength-1:0] x1r_10, x1i_10;
wire [wordlength-1:0] z2r_10, z2i_10;

wire [14:0]     out_test;
wire [14:0]     t_offset_diff,    // Actual +/- difference
           t_offset_thresh,   // Valid offset (maybe)
           t_offset_dly,      // Delayed of above.
           t_offset_scalled,  // Scalled to t_offset.
           read_pos,          // read trig, +ve offset
           read_neg,          // read trig, -ve offset
           write_pos,         // write trg, +ve offset
           write_neg;         // write trg, -ve offset

assign out_test = t_offset_diff;
// -----------------------------------------------------------------
//    Fast 40 MHz clock decoder and valid_in control.
// -----------------------------------------------------------------

always @(posedge clk)
 if (!nrst)                // Synchronous power-up reset.
  r <= 0;
 else if (valid_in)              // Count if input data valid.
  r <= r + 1'b1;

assign enable_0_4 = valid_in & (~r[1] & ~r[0]); // Gate valid_in with
assign enable_1_4 = valid_in & (~r[1] & r[0]); // decoded enable signals
assign enable_2_4 = valid_in & ( r[1] & ~r[0]); // to control all reg's.
assign enable_3_4 = valid_in & ( r[1] & r[0]); // Enables every 4 clk's

assign enable_1_8 = valid_in & (~r[2] & ~r[1] & r[0]);
assign enable_2_8 = valid_in & (~r[2] & r[1] & ~r[0]);
assign enable_3_8 = valid_in & (~r[2] & r[1] & r[0]);
assign enable_4_8 = valid_in & ( r[2] & ~r[1] & ~r[0]); // Enables every 8
assign enable_5_8 = valid_in & ( r[2] & ~r[1] & r[0]); // clk's
assign enable_6_8 = valid_in & ( r[2] & r[1] & ~r[0]);
assign enable_7_8 = valid_in & ( r[2] & r[1] & r[0]);

// -----------------------------------------------------------------
// The entire data path incorporating the FIFO's, ROM and comparators.
// -----------------------------------------------------------------

// Register the data inputs to the windowing module.
always @(posedge clk)
 if (in_resync || !nrst)
```

```
begin
  xr_reg <= in_xr;
  xi_reg <= in_xi;
end
else if (enable_3_4)
 begin
  xr_reg <= in_xr;
  xi_reg <= in_xi;
 end

// Take the modulus of in_xr and in_xi and add together (|in_xr| + |in_xi|).
always @(xr_reg or xi_reg)
 begin
  if (xr_reg[wordlength-3])      // Checking MSB for negative number.
    xr_tmp1 = -xr_reg;
  else
    xr_tmp1 = xr_reg;

  if (xi_reg[wordlength-3])      // Checking MSB for negative number.
    xi_tmp1 = -xi_reg;
  else
    xi_tmp1 = xi_reg;

  xri_tmp1 = xr_tmp1 + xi_tmp1;
 end

 assign even_symbol = r[2];

 always @(even_symbol or msb_out_tmp or ram_in or lsb_out) // Mux MSB/LSB to
  if (even_symbol)               // allow 1K RAM
   begin                    // to act as a 2K
     ram_out = lsb_out;               // FIFO, possible
     lsb_in_tmp = ram_in;              // since data
   end                     // bitwidth is 2b
  else                      // bits wide in
   begin                    // the 1K RAM and
     ram_out = msb_out_tmp;              // only b bits are
     msb_in = ram_in;               // required in the
   end                     // data path.

  always @(posedge clk)              // Delay even
   begin                    // symbols by one
    if (enable_5_8)               // symbol so that
      lsb_in <= lsb_in_tmp;             // two symbols are
    if (enable_7_8)               // written & read
      msb_out_tmp <= msb_out;              // to the ram.
   end

  assign xri_tmp2 = ram_out;             // Map RAM I/O
  assign ram_in = xri_tmp1;             // to dp wires.

  always @(ram10_out or msb_in or lsb_in or z2r_10 or z2i_10
      or ram_enable_8 or enable_3_8
         or fft_ram_enable or fft_ram_rnotw
         or window_ram_addr or fft_ram_addr
         or tracking)          // FFT/WINDOW FIFO
```

```
begin                   // RAM Mux code.
 if (!tracking)            // In window acq
   begin                 // mode.
    msb_out = ram10_out[2*wordlength-1:wordlength];
    lsb_out = ram10_out[wordlength-1:0];       // Connect window
    ram10_in[2*wordlength-1:wordlength] = msb_in;  // datapath & RAM
    ram10_in[wordlength-1:0] = lsb_in;        // control signals
    ram_enable = ram_enable_8;
    ram_rnotw = enable_3_8;
    ram_addr = window_ram_addr;
   end
 else                    // In tracking
   begin                  // mode, therefore
    x1r_10 = ram10_out[2*wordlength-1:wordlength];  // FFT functional.
    x1i_10 = ram10_out[wordlength-1:0];
    ram10_in[2*wordlength-1:wordlength] = z2r_10;  // Connect FFT
    ram10_in[wordlength-1:0] = z2i_10;        // datapath & RAM
    ram_enable = fft_ram_enable;           // control signals
    ram_rnotw = fft_ram_rnotw;
    ram_addr = fft_ram_addr;
   end
end

assign track_ram_rnotw = enable_3_4 & read;
assign track_ram_enable = (enable_3_4 & read) || (enable_1_4 & write);

// Select which FIFO we read data from depending on tracking or acquire mode.
always @(xri_tmp5 or xri_tmp2 or tracking)
 if(tracking)
  xri_tmp6 = xri_tmp5;            // Tracking mode
 else                   // data.
  xri_tmp6 = xri_tmp2;            // Acquisition
                      // mode data.
// Perform computation of s(i-j)
always @(xri_tmp1 or xri_tmp6)
 xri_tmp7 = xri_tmp1 - xri_tmp6;

// Take the modulus of xri_tmp7;
always @(xri_tmp7)
 if (xri_tmp7[wordlength-2])          // Check MSB for
  xri_tmp3 = -xri_tmp7;            // neg number.
 else
  xri_tmp3 = xri_tmp7;

// Setup FIFO to perform moving summation of s(i-j) values.
fft_sr_addr #(wordlength-2, FIFO_N) sr_N (clk, dp_control, // Length=FIFO_N.
                xri_tmp3,   // Input.
                xri_tmp4);  // Output.

// Compute the moving summation i.e S(i-j) = s(i-1,j-1) + s(i-2,j-2) + ...
// We must NOT truncate or round acc as the error will grow across a symbol.
always @(posedge clk)
 if (in_resync || !nrst || dpctl_reset)    // Clear accumulator at
  acc <= 0;               // power-up or Resync or trk.
 else if (dp_control & acc_add)        // Wait until acc data valid.
 // Subtract as well as add when 2K/8K FIFO is full.
```

```
acc <= acc + xri_tmp3 - ((acc_add_sub) ? xri_tmp4 : 0);

assign lu_address = acc;      // Ensure lu_address is large enough to
                              // accomodate acc number range.

fft_window_lu #(r_wordlength, lu_AddressSize)    // Case table instance
log_lu (clk, dp_control, lu_address, lu_data);   // for a log lookup.

// Setup 5 bit FIFO to determine the delayed variance.
fft_sr_addr #(r_wordlength, FIFO_n) sr_n (clk, dp_control, // Length=FIFO_n.
                    lu_data,    // Input.
                    xri_tmp9);  // Output.

// Determine difference of logs and hence the f_ratio when it is valid.
always @(lu_data or xri_tmp9 or f_ratio_valid)
 f_ratio = (f_ratio_valid) ? lu_data - xri_tmp9 : 1'b0;

// -----------------------------------------------------------------
//      Positive threshold (for information only)
// -----------------------------------------------------------------

assign pos_peak =((f_ratio >= pos_threshold &&
        f_ratio < (1 << r_wordlength)) ? 1'b1 : 1'b0);

// -----------------------------------------------------------------
//       FFT window datapath control registers.
// -----------------------------------------------------------------

always @(posedge clk)
 if (in_resync || !nrst || dpctl_reset)      // Synchronous reset.
  begin
   f_ratio_valid <= 1'b0;           // Initalise datapath
   acc_add <= 1'b0;                 // control registers.
   acc_add_sub <= 1'b0;
  end
 else if (enable_3_4 && ~read)          // Acquisition mode
  begin                  // Use 2K/8K FIFO.
   if (dp_count == 2047 + FIFO_N + FIFO_n + 1 + 1)  // f_ratio only valid
    f_ratio_valid <= 1'b1;           // after sum of FIFO
   if (dp_count == 2047)              // +acc+ROM latencys
    acc_add <= 1'b1;               // Add if acc full.
   if (dp_count == 2047+FIFO_N)          // Add/sub when FIFO
    acc_add_sub <= 1'b1;             // N is full.
  end
 else if (enable_3_4 && read)          // Tracking mode
  begin                  // Use FIFO_L.
   if (dp_count == FIFO_L + FIFO_N + FIFO_n + 1 + 1) // f_ratio only valid
    f_ratio_valid <= 1'b1;           // after sum of FIFO
   if (dp_count == FIFO_L)             // +acc+ROM latencys
    acc_add <= 1'b1;               // Add if acc full.
   if (dp_count == FIFO_L + FIFO_N)        // Add/sub when FIFO
    acc_add_sub <= 1'b1;             // N is full.
  end

always @(posedge clk)
 if (in_resync || !nrst)            // Synchronous reset.
```

```
  fifo_a_add_sub <= 0;
 else if (enable_3_4 && fifo_a_count == FIFO_A)   // fifo_a is full
  fifo_a_add_sub <= 1;                  // so add and sub.

always @(posedge clk)
 if (in_resync || !nrst)           // Synchronous reset.
  t_offset_avg_valid <= 1'b0;          // Average value is
 else if (enable_3_4 && fifo_a_count == FIFO_A + 1)  // valid one cycle
  t_offset_avg_valid <= 1'b1;          // after add_sub sig.

assign dp_control = enable_3_4 &&          // Datapath enable
       (~track_mode || track_mode && read); // in acq/track mode.

assign t_offset_ctl = enable_3_4 && t_offset_valid  // clock averager
        && pulse && !read && tracking;  // dp control signal.

// ----------------------------------------------------------------------
//  FFT window timing and sync acquisition/tracking timing counters.
// ----------------------------------------------------------------------

always @(posedge clk)
 if (in_resync || !nrst || t_reset)    // Synchronous power-up reset.
  t_count <= 0;               // Reset main timing counter.
 else if (enable_3_4 && t_retime_acq)     // Retime to count from last
  t_count <= t_count - guard_active;    // peak to current time.
 else if (enable_3_4 && ~track_mode)     // Count if not in track mode
  t_count <= t_count + 1'b1;
 else if (enable_3_4 && t_retime_trk)    // Otherwise must be in track
  t_count <= t_count - guard_active    // so advance timing for acq
       + (2*FIFO_N + FIFO_n + 2); // FIFO_L read trig point then
 else if (enable_3_4)
  begin              // wrap round t_count at
    if (t_count == 2047+guard_length)   // end of guard+active length.
     t_count <= 0;          // Needed as a reference to
    else               // track peak movement in
     t_count <= t_count + 1'b1;      // capture window.
  end

always @(posedge clk)
 if (in_resync || !nrst || g_a_reset)    // Synchronous power-up reset.
  g_a_count <= 0;            // Reset guard_active counter.
 else if (enable_3_4 && f_ratio_valid)    // g_a count when f_ratio vald
  g_a_count <= g_a_count + 1'b1;      // Guard active timing counter

always @(posedge clk)          // Datapath timing counter.
 if (in_resync || !nrst || dpctl_reset)   // Synchronous reset.
  dp_count <= 0;             // Reset datapath control.
 else if (enable_3_4 && ~track_mode)     // Always count in acquire
  dp_count <= dp_count + 1'b1;        // mode on clk 0.
 else if (enable_3_4 && track_mode && read)  // Count when reading data in
  dp_count <= dp_count + 1'b1;        // tracking mode.

always @(posedge clk)
 if (in_resync || !nrst)          // Synchronous reset.
  fifo_a_count <= 0;
 else if (enable_3_4 && t_offset_ctl)    // Only clock averager if Trk
```

```
fifo_a_count <= fifo_a_count + 1'b1;   // and t_offset is valid.

always @(posedge clk)          // Create pulse on entering
 if (enable_3_4)           // track 4 or track 5 to clk
  begin                // t_offset_ctl once per state
   if ((state == track1 &&        // transition. We need to
      old_state != track1) ||      // clock the averager only
     (state == track2 &&         // once on entering state 4 or
      old_state != track2))       // state 5 hence t_offset_ctl
     pulse <= 1'b1;           // is gated with pulse.
   else
     pulse <= 1'b0;
   old_state <= state;
  end

always @(posedge clk)
 if (in_resync || !nrst)
  tracking <= 1'b0;           // Read from 2K/8K FIFO first.
 else if (enable_3_4 && track_mode
     && dp_count == FIFO_L+1)    // Check if FIFO_L full in trk
  tracking <= 1'b1;           // then read tracking FIFO_L.

// -------------------------------------------------------------------
//     FFT window timing and sync acquisition/tracking FSM
// -------------------------------------------------------------------

always @(posedge clk)           // Acquisition mode FSM.
 if (in_resync || !nrst)         // Synchronous power-up reset.
  begin
    state <= start;          // FSM starts in resync.
    track_mode <= 1'b0;          // Start in acquisition mode.
    t_reset <= 1'b0;          // Reset main timing counter.
   dpctl_reset <= 1'b0;          // dp_ctl out of reset.
    g_a_reset <= 1'b0;          // Reset guard_active counter.
    max_peak <= 1'b0;          // Reset max peak value.
    retry <= 0;          // Reset no of retry's.
    acq_symbols <= 0;          // Reset acquired no symbols.
    guard_valid <= 1'b0;          // Guard data is valid.
    t_retime_acq <= 1'b0;          // Do not retime at resync.
    t_retime_trk <= 1'b0;          // Do not retime at resync.
  end
 else if (enable_3_4)
  case (state)
/*S0*/   start: begin
      g_a_reset <= 1'b0;          // g_a_reset out of rst
         t_reset <= 1'b0;          // t_count out of reset.
         guard_valid <= 1'b0;          // Guard invalid.
         // MUST ACT ON RETRYS TOO!!
         state <= peak1;          // Enter peak1 state.
        end

/*S1*/   peak1: begin
       t_reset <= 1'b0;          // t_count out of reset.
       if (g_a_count < 2048+512)      // Search for pos peak1
         begin
        if (f_ratio > max_peak &&
```

```
f_ratio < (1 << r_wordlength)) // Is new peak larger?
begin
max_peak <= f_ratio;     // If so assign max_peak
t_reset <= 1;        // Reset timing counter.
end
end
else              // First block complete.
begin
t_reset <= 1'b0;        // t_count out of reset.
g_a_reset <= 1'b1;       // Reset g_a_count.
max_peak <= 1'b0;        // Reset max peak value.
state <= peak2;        // Next block search.
end
end

/*S2*/  peak2: begin
g_a_reset <= 1'b0;          // Next block start cnt
if (g_a_count < 2048+512)      // Search for pos peak2
begin
if (f_ratio > max_peak &&
f_ratio < (1 << r_wordlength)) // Is new peak larger?
begin
max_peak <= f_ratio;      // If so assign max_peak
guard_active <= t_count;   // Assign guard_active.
end
end              // Second block complete
else if(// First, one peak per block situation (large guards)
(guard_active < (2560+delta)&&  // Test for 2048+512
guard_active > (2560-delta))|| // pt guard length.

(guard_active < (2304+delta)&&  // Test for 2048+256
guard_active > (2304-delta))|| // pt guard length.

(guard_active < (2176+delta)&&  // Test for 2048+128
guard_active > (2176-delta))|| // pt guard length.

(guard_active < (2112+delta)&&  // Test for 2048+64
guard_active > (2112-delta))|| // pt guard length.

// Now two peaks per block situation (small guards)
(guard_active < (5120+delta)&&  // Test 4096+512+512
guard_active > (5120-delta))|| // pt guard length.

(guard_active < (4608+delta)&&  // Test 4096+256+256
guard_active > (4608-delta))|| // pt guard length.

(guard_active < (4352+delta)&&  // Test 4096+128+128
guard_active > (4352-delta))|| // pt guard length.

(guard_active < (4224+delta)&&  // Test 4096+64+64
guard_active > (4224-delta)))  // pt guard length.
begin
state <= peak3;       // Next peak search.
g_a_reset <= 1'b1;      // Reset g_a_count.
max_peak <= 1'b0;      // Reset maximum peak.
guard_valid <= 1'b1;
```

```
              t_retime_acq <= 1'b1;
               end
            else              // Acquisition failed so
             begin             // jump to start and
              state <= start;     // increment the retry
              retry <= retry + 1'b1;     // counter.
              t_reset <= 1'b1;        // Reset t_count.
              g_a_reset <= 1'b1;       // Reset g_a_count.
              max_peak <= 1'b0;       // Reset maximum peak.
             end
            end

/*S3*/  peak3: begin
          t_retime_acq <= 1'b0;
            g_a_reset <= 1'b0;        // Next block start cnt
            if (g_a_count < 2048+512)      // Search for pos peak2
             begin
           if (f_ratio > max_peak &&
              f_ratio < (1 << r_wordlength)) // Is new peak larger?
                begin
               max_peak <= f_ratio;     // If so assign max_peak
               guard_active <= t_count;   // Assign guard_active.
                end
             end               // third block complete
            else if(// First, one peak per block situation (large guards)
               (guard_active < (2048+guard_length // Peak test 2048
                    +delta)&&   // + guard length.
            guard_active > (2048+guard_length
                    -delta))||

              // Now two peaks per block situation (small guards)
              (guard_active < (4096+(2*guard_length)// Peak 4096 + 2
                    +delta)&&    //*guard length.
              guard_active > (4096+(2*guard_length)
                    -delta)))
             begin
              acq_symbols <= acq_symbols+1'b1;// Another sym acqurd
                 g_a_reset <= 1'b1;      // Reset g_a_count.
              max_peak <= 1'b0;      // Reset maximum peak.
              t_retime_trk <= 1'b1;    // Retime t_count to trk
              track_mode <= 1'b1;     // Enter track mode.
            dpctl_reset <= 1'b1;     // Reset datapath count
                 state <= track1;      // Enter track1 state.
               end
             else              // Acquisition failed so
              begin            // jump to start and
               state <= start;      // increment the retry
               retry <= retry + 1'b1;     // counter.
               t_reset <= 1'b1;        // Reset t_count.
               g_a_reset <= 1'b1;       // Reset g_a_count.
               max_peak <= 1'b0;       // Reset maximum peak.
              end
           end

 /*S4*/ track1: begin
             t_retime_trk <= 1'b0;       // t_count out retime.
```

```
          dpctl_reset <= 1'b0;        // dp ctl out of reset.
          if (read && f_ratio_valid)     // Peak detect on rd&vld
           begin
        if (f_ratio > max_peak &&
            f_ratio < (1 << r_wordlength)) // Is new peak larger?
            begin
            max_peak <= f_ratio;    // If so assign max_peak
            t_offset <= t_count;    // Store peak offset.
              end
            if (read_address == FIFO_L-1)  // If at end of FIFO_L
           begin            // move to next state.
            state <= track2;      // (read_Addr <> FIFO_L)
            max_peak <= 1'b0;      // Reset max peak value.
           end
           end
          else
           state <= track1;         // else wait in track1.
          end

/*S5*/ track2: begin
           if (read && f_ratio_valid)      // Peak detect on rd&vld
            begin
        if (f_ratio > max_peak &&
            f_ratio < (1 << r_wordlength)) // Is new peak larger?
            begin
            max_peak <= f_ratio;    // If so assign max_peak
            t_offset <= t_count;    // Store peak offset
              end
            if (read_address == FIFO_L-1)  // At end of FIFO_L
           begin            // move to next state.
            state <= track1;      // (read_Addr <> FIFO_L)
            max_peak <= 1'b0;      // Reset max peak value.
           end
           end
          else
           state <= track2;         // Wait in this state.
          end

      default:  state <= 3'bXXX;
    endcase

    // ------------------------------------------------------------------
    //         FFT window output decode logic.
    // ------------------------------------------------------------------

    always @(posedge clk)
     if (in_resync || !nrst)          // Synchronous reset.
      out_iqgi <= 0;
     else if (enable_3_4 && tracking &&
         t_count == 15'd0 - iqdemod_latency)   // iqgi guard start.
      out_iqgi <= 1'b1;
     else if (enable_3_4 && tracking &&
              t_count == iqdemod_latency) // iqgi guard stop.
      out_iqgi <= 1'b0;

    always @(posedge clk)
```

```
if (in_resync || !nrst)             // Synchronous reset.
 out_sincgi <= 0;
else if (enable_3_4 && tracking &&
    t_count == 15'd0 - sincint_latency)   // sincgi guard start.
 out_sincgi <= 1'b1;
else if (enable_3_4 && tracking && // TO COMPLETE LATENCY STUFF
        t_count == sincint_latency) // sincgi guard stop.
 out_sincgi <= 1'b0;

always @(posedge clk)               // Count over active
 if (in_resync || !nrst)            // interval to generate
  enable_fft <= 1'b0;              // FFT valid pulse.
 else if (enable_3_4 && tracking &&
  t_count == guard_length + FIFO_L/2 - w_advance) // FFT start point is
  enable_fft <= 1'b1;              // in middle of write
 else if (enable_3_4 && tracking &&        // into FIFO_L + advced.
        fft_valid_count == 2047) // FFT stop after 2048
  enable_fft <= 1'b0;              // samples.

always @(posedge clk)
 if (in_resync || !nrst)            // Synchronous reset.
  fft_valid_count <= 0;
 else if (enable_3_4 && tracking && ~enable_fft)  // Valid count = 0.
  fft_valid_count <= 0;            // until fft is enabled.
 else if (enable_3_4 && tracking && enable_fft)
  fft_valid_count <= fft_valid_count + 1'b1;   // Count when enabled.

assign valid_out = enable_fft & valid_in;  //MUST SYNCHROS Vld every 3 clks?

// ----------------------------------------------------------------------
//        Synchronous RAM address generators.
// ----------------------------------------------------------------------

always @(posedge clk)              // Acqsition FIFO address gen.
 if (!nrst || in_resync)          // Synchronous reset.
  window_ram_addr <= 0;           // Address gen for acq mode.
 else if (enable_2_8)
  window_ram_addr <= window_ram_addr + 1'b1;

assign ram_enable_8 = enable_2_8 || enable_3_8 ||
        enable_4_8 || enable_5_8;

always @(posedge clk)              // Tracking FIFO address gen.
 begin
  if (!nrst || in_resync)
    begin
     read_address <= 0;          // Reset track FIFO read addr.
     write_address <= 0;         // Reset track FIFO write addr
     write <= 1'b0;             // Track FIFO, write disabled.
     read <= 1'b0;              // Track FIFO, read disabled.
    end
  else if (enable_3_4)
    begin
     if (track_mode && t_count == 0)      // Track FIFO read
      read <= 1'b1;              // trigger point.
```

```
    if (read)                    // Read if `read'
     begin                      // flag is set.
      if (read_address == FIFO_L-1)        // Stop read at
         begin                 // end of FIFO.
          read_address <= 0;
          read <= 1'b0;              // Clr read flag.
         end
      else
       read_address <= read_address + 1'b1;     // Inc r address.
     end

   if (track_mode && t_count == guard_length+1)   // Write if the
     write <= 1'b1;                  // read is guard
                              // depth into FIFO
   if (write)
     begin
      if (write_address == FIFO_L-1)        // Stop write at
         begin                 // end of FIFO.
          write_address <= 0;
          write <= 1'b0;
         end
       else
         write_address <= write_address + 1'b1;    // Inc w address.
    end
    end
  end

  always @(enable_1_4 or enable_3_4 or read or write or  // Assign read and
     read_address or write_address)          // write addresses
  if (enable_3_4 && read)                // onto common
   track_ram_address = read_address;           // address bus
  else if (enable_1_4 && write)             // for tracking
   track_ram_address = write_address;          // tsyncram RAM.

  // -------------------------------------------------------------------
  //   Thresholding function to determine precise guard interval.
  // -------------------------------------------------------------------

  always @(posedge clk)
   if (enable_3_4 && guard_valid)
    begin
      // First, one peak per block situation (large guards)
      if (guard_active < (2560+delta)&&      // Test for 2048+512
        guard_active > (2560-delta))        // pt guard length.
       begin
        out_rx_guard <= 2'b11;
        guard_length <= 512;
       end

      if (guard_active < (2304+delta)&&       // Test for 2048+256
        guard_active > (2304-delta))        // pt guard length.
       begin
        out_rx_guard <= 2'b10;
        guard_length <= 256;
       end
```

```
if (guard_active < (2176+delta)&&      // Test for 2048+128
  guard_active > (2176-delta))        // pt guard length.
 begin
  out_rx_guard <= 2'b01;
  guard_length <= 128;
 end

if (guard_active < (2112+delta)&&      // Test for 2048+64
  guard_active > (2112-delta))        // pt guard length.
 begin
  out_rx_guard <= 2'b00;
  guard_length <= 64;
 end

// Now two peaks per block situation (small guards)
if (guard_active < (5120+delta)&&      // Test for 4096+512+512
  guard_active > (5120-delta))        // 512 pt guard length.
 begin
  out_rx_guard <= 2'b11;
  guard_length <= 512;
 end

if (guard_active < (4608+delta)&&      // Test for 4096+256+256
  guard_active > (4608-delta))        // 256 pt guard length.
 begin
  out_rx_guard <= 2'b10;
  guard_length <= 256;
 end

if (guard_active < (4352+delta)&&      // Test for 4096+128+128
  guard_active > (4352-delta))        // 128 pt guard length.
 begin
  out_rx_guard <= 2'b01;
  guard_length <= 128;
 end

if (guard_active < (4224+delta)&&      // Test for 4096+64+64
  guard_active > (4224-delta))        // 64 pt guard length.
 begin
  out_rx_guard <= 2'b00;
  guard_length <= 64;
 end
end

// ---------------------------------------------------------------
//       Averager for t_offset in tracking mode.
// ---------------------------------------------------------------

assign t_offset_diff = t_offset - (2*FIFO_N + FIFO_n); //dly 2 for latency?

always @(posedge clk)
 if (in_resync || !nrst) //NEED TO ENABLE THIS!!!!!!
  t_offset_valid <= 0;
 else if ((t_offset_diff < (1 << 14 + 1) - t_offset_threshold && // Neg
     t_offset_diff > (1 << 14 - 1)) ||
     (t_offset_diff > t_offset_threshold &&       // Pos
```

```
      t_offset_diff < (1 << 14)) //CORRECT TO DETECT vld = 1 not 0
     )
  t_offset_valid <= 0;
 else
  t_offset_valid <= 1;

assign t_offset_thresh = (t_offset_valid) ? t_offset_diff : 0;

// Setup FIFO to perform moving summation of t_offset values.
fft_sr_addr #(15, FIFO_A) sr_A (clk, t_offset_ctl,
                   t_offset_thresh,     // Input.
                   t_offset_dly);      // Output.

// Compute the moving summation i.e t_offset(i-1) + t_offset(i-2) + ...
// We must NOT truncate or round acc as the error will grow across a symbol.
always @(posedge clk)
 if (in_resync || !nrst)          // Clear accumulator at
  t_offset_avg <= 0;              // power-up or Resync.
 else if (t_offset_ctl)           // Wait until t_offset valid.
  // Subtract as well as add when averager is full.
  t_offset_avg <= t_offset_avg + t_offset_thresh
          - ((fifo_a_add_sub) ? t_offset_dly : 0);

assign t_offset_scalled =
   {{(FIFO_A_bits){t_offset_avg[14]}},t_offset_avg[14:FIFO_A_bits]};

// ---------------------------------------------------------------
// Code to determine conditions for advancing/retarding tracking window.
// ---------------------------------------------------------------

assign read_pos = t_offset_scalled;          // +ve (late) so
                    // delay read

assign read_neg = 2047 + guard_length + 1 -       // -ve (early) so
       (~t_offset_scalled + 1);          // advance read

assign write_pos = guard_length + 1 +          // +ve (late) so
       t_offset_scalled;           // delay write

// PROBLEMS WHEN offset > guard_length + 1
// (should not happen as we range check peaks in acq mode)
assign write_neg = guard_length + 1 -          // -ve (early) so
       (~t_offset_scalled + 1);          // advance write

endmodule
```

Listing 15

```
// SccsId: %W% %G%
/******************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

  Author   : Dawood Alam.

  Description: Verilog code for a structural netlist coupling the Fast Fourier
     Transform (FFT) processor to the window acquisition hardware.
```

Notes :

```
*****************************************************************************/

`timescale 1ns / 100ps

module fft_top      (i_data,
            q_data,
              clk,
              nrst,
              in_resync,
              in_2k8k,
              valid_in,
              ram4_in,
            ram5_in,
            ram6_in,
            ram7_in,
            ram8_in,
            ram9_in,
                ram10_in,
            i_out,
              q_out,
              out_ovf,
            enable_0,
              enable_1,
              enable_2,
              enable_3,
            valid_out,
              ram4_out,
            ram5_out,
              ram6_out,
            ram7_out,
              ram8_out,
              ram9_out,
              ram10_out,
              ram_addr,
              ram_enable,
              ram_rnotw,
              rom3_addr,
              rom4_addr,
              rom3_data,
              rom4_data,
              track_addr,
              track_data_in,
            track_data_out,
              track_rnw,
            track_ram_enable,
              out_rx_guard,
              out_iqgi,
            out_sincgi,
              out_test);

    // -----------------------------------------------------------------------
    //          Parameter definitions.
    // -----------------------------------------------------------------------
```

```
parameter       wordlength = 12;     // Data wordlength.
parameter       c_wordlength = 10;   // Coeff wordlength.
parameter       AddressSize = 13;    // Size of address bus.
parameter       rom_AddressSize = 13;  // ROM address bus size.
parameter       mult_scale = 3;      // Multiplier scalling:
                 // 1 = /4096, 2 = /2048,
                 // 3 = /1024, 4 = /512.
parameter       r_wordlength = 10;   // ROM data wordlength.
parameter       FIFO_L = 256;        // Tracking FIFO length.
parameter       FIFO_L_bits = 8;     // Track FIFO addr bits
parameter       FIFO_N = 64;         // Acc length S(i-j).
parameter       FIFO_n = 64;         // Acc length S(i-n-j).
parameter       FIFO_A = 32;         // t_offset delay FIFO.
parameter       FIFO_A_bits = 5;     // Track FIFO bits.
parameter       lu_AddressSize = 15;  // log rom address size.
parameter       delta = 20;          // Gu threshold distance
parameter       acquired_symbols = 2;  // Acq symbls before trk
parameter       pos_threshold = 3;   // for info only.
parameter       t_offset_threshold = 10; // t_offset valid thresh
parameter       w_advance = 10;      // win trig frm boundary
parameter       sincint_latency = 2;  // Latency to sinc intep
parameter       iqdemod_latency = 168;  // Latency to IQ demod.

// -----------------------------------------------------------------
//          Input/Output ports.
// -----------------------------------------------------------------

input        clk,          // Master clock.
          nrst,          // Power-up reset.
            in_2k8k,        // 2K mode active low.
            valid_in,       // Input data valid.
            in_resync;

input [9:0]      i_data,          // FFT input data, I.
          q_data;        // FFT input data, Q.

input [wordlength-3:0]  track_data_out;

input [wordlength*2-1:0] ram4_out,        // Couple the I/Q data
          ram5_out,        // outputs from the
            ram6_out,        // memory to the
          ram7_out,        // respective butterfly
            ram8_out,        // processors.
            ram9_out,
            ram10_out;

input [c_wordlength*2-1:0] rom3_data,
          rom4_data;

output [rom_AddressSize-6:0] rom3_addr;
output [rom_AddressSize-4:0] rom4_addr;

output [14:0]     out_test;        // Temp testpin output.

output [1:0]      out_rx_guard;       // Acquired gu length.
```

```
output [wordlength-3:0]  track_data_in;

output [wordlength*2-1:0] ram4_in,        // Couple the I/Q data
        ram5_in,        // outputs of each BF
        ram6_in,        // processor to their
        ram7_in,        // respective memory
        ram8_in,        // inputs.
        ram9_in,
        ram10_in;

output [AddressSize-1:0]  ram_addr;            // RAM address bus.

output        out_ovf,        // Overflow flag.
        enable_0,        // Enable clock 0.
        enable_1,        // Enable clock 1.
        enable_2,        // Enable clock 2.
        enable_3,        // Enable clock 3.
        valid_out,        // Output data valid.
        ram_enable,        // RAM enable.
        ram_rnotw,
        track_rnw,
        track_ram_enable,
        out_iqgi,
        out_sincgi;

output [FIFO_L_bits-1:0]  track_addr;

output [wordlength-1:0]  i_out,        // FFT output data, I.
        q_out;        // FFT output data, Q.

// ---------------------------------------------------------------------
//        Wire/register declarations.
// ---------------------------------------------------------------------

wire  [9:0]        i_data,        // FFT/WIN input I.
        q_data;        // FFT/WIN output Q.

wire  [wordlength-1:0]  i_out,        // FFT output data, I.
        q_out;        // FFT output data, Q.

wire  [wordlength*2-1:0] ram4_in,
        ram5_in,
        ram6_in,
        ram7_in,
        ram8_in,
        ram9_in,
        ram10_in;

wire  [wordlength*2-1:0] ram4_out,
        ram5_out,
        ram6_out,
        ram7_out,
        ram8_out,
        ram9_out,
        ram10_out;
```

```
wire [AddressSize-1:0] ram_addr,        // RAM address bus.
        ram_addr_fft_2_win;

wire        clk,
        nrst,
          in_2k8k,
        in_resync,
        valid_in,
          out_ovf,
          enable_0,
          enable_1,
          enable_2,
          enable_3,
        valid_out;
        ram_enable,       // RAM enable signal.
          ram_rnotw,
        valid_win_2_fft,
          ram_rnotw_fft_2_win,
          ram_enable_fft_2_win,
        track_rnw,
        track_ram_enable,
          out_iqgi,
        out_sincgi;

wire [wordlength-1:0] x1r_10, x1i_10,
        z2r_10, z2i_10;

wire [wordlength-3:0] track_data_in,
        track_data_out;

wire [FIFO_L_bits-1:0] track_addr;

wire [1:0]      out_rx_guard;      // Determined guard.

wire [c_wordlength*2-1:0] rom3_data,
        rom4_data;

wire [rom_AddressSize-6:0] rom3_addr;
wire [rom_AddressSize-4:0] rom4_addr;

wire [14:0]      out_test;

// ------------------------------------------------------------------
//        Instance FFT processor.
// ------------------------------------------------------------------

fft_r22sdf      #(wordlength,
        c_wordlength,
          AddressSize,
        rom_AddressSize,
          mult_scale)
        fft (.in_xr(i_data),       // FFT input data, I.
        .in_xi(q_data),      // FFT input data, Q
        .clk(clk),        // Master clock.
        .nrst(nrst),        // Power-up reset.
          .in_2k8k(in_2k8k),      // 2K active low.
```

```
.valid_in(valid_win_2_fft),// Input valid.
.out_xr(i_out),     // FFT output data, I.
.out_xi(q_out),     // FFT output data, Q.
.out_ovf(out_ovf),  // Overflow flag.
.enable_0(enable_0),
.enable_1(enable_1),
.enable_2(enable_2),
.enable_3(ram_rnotw_fft_2_win),
.valid_out(valid_out),
.ram_address(ram_addr_fft_2_win),
.ram_enable(ram_enable_fft_2_win),
.address_rom3(rom3_addr),
.address_rom4(rom4_addr),

// RAM input ports.
.z2r_4(ram4_in[wordlength-1:0]),
.z2i_4(ram4_in[wordlength*2-1:wordlength]),
.z2r_5(ram5_in[wordlength-1:0]),
.z2i_5(ram5_in[wordlength*2-1:wordlength]),
.z2r_6(ram6_in[wordlength-1:0]),
.z2i_6(ram6_in[wordlength*2-1:wordlength]),
.z2r_7(ram7_in[wordlength-1:0]),
.z2i_7(ram7_in[wordlength*2-1:wordlength]),
.z2r_8(ram8_in[wordlength-1:0]),
.z2i_8(ram8_in[wordlength*2-1:wordlength]),
.z2r_9(ram9_in[wordlength-1:0]),
.z2i_9(ram9_in[wordlength*2-1:wordlength]),
.z2r_10(z2r_10),// Frm FFT datapath to window (I).
.z2i_10(z2i_10),// Frm FFT datapath to window (Q).

// RAM output ports.
.x1r_4(ram4_out[wordlength-1:0]),
.x1i_4(ram4_out[wordlength*2-1:wordlength]),
.x1r_5(ram5_out[wordlength-1:0]),
.x1i_5(ram5_out[wordlength*2-1:wordlength]),
.x1r_6(ram6_out[wordlength-1:0]),
.x1i_6(ram6_out[wordlength*2-1:wordlength]),
.x1r_7(ram7_out[wordlength-1:0]),
.x1i_7(ram7_out[wordlength*2-1:wordlength]),
.x1r_8(ram8_out[wordlength-1:0]),
.x1i_8(ram8_out[wordlength*2-1:wordlength]),
.x1r_9(ram9_out[wordlength-1:0]),
.x1i_9(ram9_out[wordlength*2-1:wordlength]),
.x1r_10(x1r_10),// To FFT datapath frm window (I).
.x1i_10(x1i_10),// To FFT datapath frm window (Q).

// ROM output ports.
.br_3(rom3_data[c_wordlength*2-1:c_wordlength]),
.bi_3(rom3_data[c_wordlength-1:0]),
.br_4(rom4_data[c_wordlength*2-1:c_wordlength]),
.bi_4(rom4_data[c_wordlength-1:0]));

// -----------------------------------------------------------------
//        Instance FFT window processor.
// -----------------------------------------------------------------
```

```
fft_window     #(wordlength,
         r_wordlength,
          AddressSize,
         FIFO_L,
         FIFO_L_bits,
         FIFO_N,
         FIFO_n,
         FIFO_A,
         FIFO_A_bits,
         lu_AddressSize,
         delta,
         acquired_symbols,
         pos_threshold,
         t_offset_threshold,
          w_advance,
          sincint_latency,
          iqdemod_latency)
       window (.in_xr(i_data),
         .in_xi(q_data),
         .clk(clk),
         .nrst(nrst),
         .valid_in(valid_in),
         .valid_out(valid_win_2_fft),
         .in_resync(in_resync),
         .out_iqgi(out_iqgi),
         .out_sincgi(out_sincgi),
         .out_rx_guard(out_rx_guard),
         .out_acquired(out_acquired),
         .out_fft_window(out_fft_window),
         .enable_3_4(enable_3),
         .out_test(out_test),
         .track_ram_address(track_addr),
         .xri_tmp1(track_data_in),
         .xri_tmp5(track_data_out),
         .track_ram_rnotw(track_rnw),
         .track_ram_enable(track_ram_enable),
          .ram_addr(ram_addr),
            .ram_enable(ram_enable),
            .ram_rnotw(ram_rnotw),
          .ram10_in(ram10_in),   // To 1K x 24 bit RAM.
            .ram10_out(ram10_out), // From 1K x 24 bit RAM.
            .x1r_10(x1r_10),     // To FFT datapath (I).
            .x1i_10(x1i_10),     // To FFT datapath (Q).
            .z2r_10(z2r_10),     // From FFT datapath (I)
            .z2i_10(z2i_10),     // From FFT datapath (Q)
          .fft_ram_rnotw(ram_rnotw_fft_2_win),
          .fft_ram_enable(ram_enable_fft_2_win),
          .fft_ram_addr(ram_addr_fft_2_win));
endmodule
```

Listing 16

```
// 2048 point FFT twiddle factor coefficients (Radix 4+2).
// Coefficients stored as non-fractional 10 bit integers (scale 1 ).
// Real Coefficient (cosine value) is coefficient high-byte.
// Imaginary Coefficient (sine value) is coefficient low-byte.
```

```
0111111111_0000000000 // W0000_2048 = +1.000000 -0.000000
0111111111_1111111110 // W0001_2048 = +0.999995 -0.003068
0111111111_1111111101 // W0002_2048 = +0.999981 -0.006136
0111111111_1111111011 // W0003_2048 = +0.999958 -0.009204
0111111111_1111111010 // W0004_2048 = +0.999925 -0.012272
0111111111_1111111000 // W0005_2048 = +0.999882 -0.015339
0111111111_1111110111 // W0006_2048 = +0.999831 -0.018407
0111111111_1111110101 // W0007_2048 = +0.999769 -0.021474
0111111111_1111110011 // W0008_2048 = +0.999699 -0.024541
0111111111_1111110010 // W0009_2048 = +0.999619 -0.027608
0111111111_1111110000 // W0010_2048 = +0.999529 -0.030675
0111111111_1111101111 // W0011_2048 = +0.999431 -0.033741
0111111111_1111101101 // W0012_2048 = +0.999322 -0.036807
0111111111_1111101100 // W0013_2048 = +0.999205 -0.039873
0111111111_1111101010 // W0014_2048 = +0.999078 -0.042938
0111111111_1111101000 // W0015_2048 = +0.998941 -0.046003
0111111111_1111100111 // W0016_2048 = +0.998795 -0.049068
0111111111_1111100101 // W0017_2048 = +0.998640 -0.052132
0111111111_1111100100 // W0018_2048 = +0.998476 -0.055195
0111111111_1111100010 // W0019_2048 = +0.998302 -0.058258
0111111111_1111100001 // W0020_2048 = +0.998118 -0.061321
0111111111_1111011111 // W0021_2048 = +0.997925 -0.064383
0111111111_1111011101 // W0022_2048 = +0.997723 -0.067444
0111111111_1111011100 // W0023_2048 = +0.997511 -0.070505
0111111111_1111011010 // W0024_2048 = +0.997290 -0.073565
0111111110_1111011001 // W0025_2048 = +0.997060 -0.076624
0111111110_1111010111 // W0026_2048 = +0.996820 -0.079682
0111111110_1111010110 // W0027_2048 = +0.996571 -0.082740
0111111110_1111010100 // W0028_2048 = +0.996313 -0.085797
0111111110_1111010011 // W0029_2048 = +0.996045 -0.088854
0111111110_1111010001 // W0030_2048 = +0.995767 -0.091909
0111111110_1111001111 // W0031_2048 = +0.995481 -0.094963
0111111110_1111001110 // W0032_2048 = +0.995185 -0.098017
0111111101_1111001100 // W0033_2048 = +0.994879 -0.101070
0111111101_1111001011 // W0034_2048 = +0.994565 -0.104122
0111111101_1111001001 // W0035_2048 = +0.994240 -0.107172
0111111101_1111001000 // W0036_2048 = +0.993907 -0.110222
0111111101_1111000110 // W0037_2048 = +0.993564 -0.113271
0111111101_1111000100 // W0038_2048 = +0.993212 -0.116319
0111111100_1111000011 // W0039_2048 = +0.992850 -0.119365
0111111100_1111000001 // W0040_2048 = +0.992480 -0.122411
0111111100_1111000000 // W0041_2048 = +0.992099 -0.125455
0111111100_1110111110 // W0042_2048 = +0.991710 -0.128498
0111111100_1110111101 // W0043_2048 = +0.991311 -0.131540
0111111011_1110111011 // W0044_2048 = +0.990903 -0.134581
0111111011_1110111010 // W0045_2048 = +0.990485 -0.137620
0111111011_1110111000 // W0046_2048 = +0.990058 -0.140658
0111111011_1110110110 // W0047_2048 = +0.989622 -0.143695
0111111010_1110110101 // W0048_2048 = +0.989177 -0.146730
0111111010_1110110011 // W0049_2048 = +0.988722 -0.149765
0111111010_1110110010 // W0050_2048 = +0.988258 -0.152797
0111111010_1110110000 // W0051_2048 = +0.987784 -0.155828
0111111001_1110101111 // W0052_2048 = +0.987301 -0.158858
0111111001_1110101101 // W0053_2048 = +0.986809 -0.161886
0111111001_1110101100 // W0054_2048 = +0.986308 -0.164913
0111111001_1110101010 // W0055_2048 = +0.985798 -0.167938
```

```
0111111000_1110101000  // W0056_2048 = +0.985278  -0.170962
0111111000_1110100111  // W0057_2048 = +0.984749  -0.173984
0111111000_1110100101  // W0058_2048 = +0.984210  -0.177004
0111111000_1110100100  // W0059_2048 = +0.983662  -0.180023
0111110111_1110100010  // W0060_2048 = +0.983105  -0.183040
0111110111_1110100001  // W0061_2048 = +0.982539  -0.186055
0111110111_1110011111  // W0062_2048 = +0.981964  -0.189069
0111110110_1110011110  // W0063_2048 = +0.981379  -0.192080
0111110110_1110011100  // W0064_2048 = +0.980785  -0.195090
0111110110_1110011011  // W0065_2048 = +0.980182  -0.198098
0111110110_1110011001  // W0066_2048 = +0.979570  -0.201105
0111110101_1110010111  // W0067_2048 = +0.978948  -0.204109
0111110101_1110010110  // W0068_2048 = +0.978317  -0.207111
0111110101_1110010100  // W0069_2048 = +0.977677  -0.210112
0111110100_1110010011  // W0070_2048 = +0.977028  -0.213110
0111110100_1110010001  // W0071_2048 = +0.976370  -0.216107
0111110100_1110010000  // W0072_2048 = +0.975702  -0.219101
0111110011_1110001110  // W0073_2048 = +0.975025  -0.222094
0111110011_1110001101  // W0074_2048 = +0.974339  -0.225084
0111110011_1110001011  // W0075_2048 = +0.973644  -0.228072
0111110010_1110001010  // W0076_2048 = +0.972940  -0.231058
0111110010_1110001000  // W0077_2048 = +0.972226  -0.234042
0111110001_1110000111  // W0078_2048 = +0.971504  -0.237024
0111110001_1110000101  // W0079_2048 = +0.970772  -0.240003
0111110001_1110000100  // W0080_2048 = +0.970031  -0.242980
0111110000_1110000010  // W0081_2048 = +0.969281  -0.245955
0111110000_1110000001  // W0082_2048 = +0.968522  -0.248928
0111101111_1101111111  // W0083_2048 = +0.967754  -0.251898
0111101111_1101111110  // W0084_2048 = +0.966976  -0.254866
0111101111_1101111100  // W0085_2048 = +0.966190  -0.257831
0111101110_1101111010  // W0086_2048 = +0.965394  -0.260794
0111101110_1101111001  // W0087_2048 = +0.964590  -0.263755
0111101101_1101110111  // W0088_2048 = +0.963776  -0.266713
0111101101_1101110110  // W0089_2048 = +0.962953  -0.269668
0111101101_1101110100  // W0090_2048 = +0.962121  -0.272621
0111101100_1101110011  // W0091_2048 = +0.961280  -0.275572
0111101100_1101110001  // W0092_2048 = +0.960431  -0.278520
0111101011_1101110000  // W0093_2048 = +0.959572  -0.281465
0111101011_1101101110  // W0094_2048 = +0.958703  -0.284408
0111101010_1101101101  // W0095_2048 = +0.957826  -0.287347
0111101010_1101101011  // W0096_2048 = +0.956940  -0.290285
0111101001_1101101010  // W0097_2048 = +0.956045  -0.293219
0111101001_1101101000  // W0098_2048 = +0.955141  -0.296151
0111101001_1101100111  // W0099_2048 = +0.954228  -0.299080
0111101000_1101100101  // W0100_2048 = +0.953306  -0.302006
0111101000_1101100100  // W0101_2048 = +0.952375  -0.304929
0111100111_1101100010  // W0102_2048 = +0.951435  -0.307850
0111100111_1101100001  // W0103_2048 = +0.950486  -0.310767
0111100110_1101011111  // W0104_2048 = +0.949528  -0.313682
0111100110_1101011110  // W0105_2048 = +0.948561  -0.316593
0111100101_1101011100  // W0106_2048 = +0.947586  -0.319502
0111100101_1101011011  // W0107_2048 = +0.946601  -0.322408
0111100100_1101011001  // W0108_2048 = +0.945607  -0.325310
0111100100_1101011000  // W0109_2048 = +0.944605  -0.328210
0111100011_1101010110  // W0110_2048 = +0.943593  -0.331106
0111100011_1101010101  // W0111_2048 = +0.942573  -0.334000
```

0111100010_1101010100	// W0112_2048 = +0.941544	-0.336890
0111100010_1101010010	// W0113_2048 = +0.940506	-0.339777
0111100001_1101010001	// W0114_2048 = +0.939459	-0.342661
0111100000_1101001111	// W0115_2048 = +0.938404	-0.345541
0111100000_1101001110	// W0116_2048 = +0.937339	-0.348419
0111011111_1101001100	// W0117_2048 = +0.936266	-0.351293
0111011111_1101001011	// W0118_2048 = +0.935184	-0.354164
0111011110_1101001001	// W0119_2048 = +0.934093	-0.357031
0111011110_1101001000	// W0120_2048 = +0.932993	-0.359895
0111011101_1101000110	// W0121_2048 = +0.931884	-0.362756
0111011101_1101000101	// W0122_2048 = +0.930767	-0.365613
0111011100_1101000011	// W0123_2048 = +0.929641	-0.368467
0111011011_1101000010	// W0124_2048 = +0.928506	-0.371317
0111011011_1101000000	// W0125_2048 = +0.927363	-0.374164
0111011010_1100111111	// W0126_2048 = +0.926210	-0.377007
0111011010_1100111110	// W0127_2048 = +0.925049	-0.379847
0111011001_1100111100	// W0128_2048 = +0.923880	-0.382683
0111011000_1100111011	// W0129_2048 = +0.922701	-0.385516
0111011000_1100111001	// W0130_2048 = +0.921514	-0.388345
0111010111_1100111000	// W0131_2048 = +0.920318	-0.391170
0111010111_1100110110	// W0132_2048 = +0.919114	-0.393992
0111010110_1100110101	// W0133_2048 = +0.917901	-0.396810
0111010101_1100110011	// W0134_2048 = +0.916679	-0.399624
0111010101_1100110010	// W0135_2048 = +0.915449	-0.402435
0111010100_1100110001	// W0136_2048 = +0.914210	-0.405241
0111010011_1100101111	// W0137_2048 = +0.912962	-0.408044
0111010011_1100101110	// W0138_2048 = +0.911706	-0.410843
0111010010_1100101100	// W0139_2048 = +0.910441	-0.413638
0111010001_1100101011	// W0140_2048 = +0.909168	-0.416430
0111010001_1100101001	// W0141_2048 = +0.907886	-0.419217
0111010000_1100101000	// W0142_2048 = +0.906596	-0.422000
0111010000_1100100111	// W0143_2048 = +0.905297	-0.424780
0111001111_1100100101	// W0144_2048 = +0.903989	-0.427555
0111001110_1100100100	// W0145_2048 = +0.902673	-0.430326
0111001101_1100100010	// W0146_2048 = +0.901349	-0.433094
0111001101_1100100001	// W0147_2048 = +0.900016	-0.435857
0111001100_1100011111	// W0148_2048 = +0.898674	-0.438616
0111001011_1100011110	// W0149_2048 = +0.897325	-0.441371
0111001011_1100011101	// W0150_2048 = +0.895966	-0.444122
0111001010_1100011011	// W0151_2048 = +0.894599	-0.446869
0111001001_1100011010	// W0152_2048 = +0.893224	-0.449611
0111001001_1100011000	// W0153_2048 = +0.891841	-0.452350
0111001000_1100010111	// W0154_2048 = +0.890449	-0.455084
0111000111_1100010110	// W0155_2048 = +0.889048	-0.457813
0111000110_1100010100	// W0156_2048 = +0.887640	-0.460539
0111000110_1100010011	// W0157_2048 = +0.886223	-0.463260
0111000101_1100010001	// W0158_2048 = +0.884797	-0.465976
0111000100_1100010000	// W0159_2048 = +0.883363	-0.468689
0111000100_1100001111	// W0160_2048 = +0.881921	-0.471397
0111000011_1100001101	// W0161_2048 = +0.880471	-0.474100
0111000010_1100001100	// W0162_2048 = +0.879012	-0.476799
0111000001_1100001010	// W0163_2048 = +0.877545	-0.479494
0111000001_1100001001	// W0164_2048 = +0.876070	-0.482184
0111000000_1100001000	// W0165_2048 = +0.874587	-0.484869
0110111111_1100000110	// W0166_2048 = +0.873095	-0.487550
0110111110_1100000101	// W0167_2048 = +0.871595	-0.490226

```
0110111101_1100000100 // W0168_2048 = +0.870087 -0.492898
0110111101_1100000010 // W0169_2048 = +0.868571 -0.495565
0110111100_1100000001 // W0170_2048 = +0.867046 -0.498228
0110111011_1100000000 // W0171_2048 = +0.865514 -0.500885
0110111010_1011111110 // W0172_2048 = +0.863973 -0.503538
0110111010_1011111101 // W0173_2048 = +0.862424 -0.506187
0110111001_1011111011 // W0174_2048 = +0.860867 -0.508830
0110111000_1011111010 // W0175_2048 = +0.859302 -0.511469
0110110111_1011111001 // W0176_2048 = +0.857729 -0.514103
0110110110_1011110111 // W0177_2048 = +0.856147 -0.516732
0110110110_1011110110 // W0178_2048 = +0.854558 -0.519356
0110110101_1011110101 // W0179_2048 = +0.852961 -0.521975
0110110100_1011110011 // W0180_2048 = +0.851355 -0.524590
0110110011_1011110010 // W0181_2048 = +0.849742 -0.527199
0110110010_1011110001 // W0182_2048 = +0.848120 -0.529804
0110110001_1011101111 // W0183_2048 = +0.846491 -0.532403
0110110001_1011101110 // W0184_2048 = +0.844854 -0.534998
0110110000_1011101101 // W0185_2048 = +0.843208 -0.537587
0110101111_1011101011 // W0186_2048 = +0.841555 -0.540171
0110101110_1011101010 // W0187_2048 = +0.839894 -0.542751
0110101101_1011101001 // W0188_2048 = +0.838225 -0.545325
0110101100_1011100111 // W0189_2048 = +0.836548 -0.547894
0110101011_1011100110 // W0190_2048 = +0.834863 -0.550458
0110101011_1011100101 // W0191_2048 = +0.833170 -0.553017
0110101010_1011100100 // W0192_2048 = +0.831470 -0.555570
0110101001_1011100010 // W0193_2048 = +0.829761 -0.558119
0110101000_1011100001 // W0194_2048 = +0.828045 -0.560662
0110100111_1011100000 // W0195_2048 = +0.826321 -0.563199
0110100110_1011011110 // W0196_2048 = +0.824589 -0.565732
0110100101_1011011101 // W0197_2048 = +0.822850 -0.568259
0110100100_1011011100 // W0198_2048 = +0.821103 -0.570781
0110100100_1011011010 // W0199_2048 = +0.819348 -0.573297
0110100011_1011011001 // W0200_2048 = +0.817585 -0.575808
0110100010_1011011000 // W0201_2048 = +0.815814 -0.578314
0110100001_1011010111 // W0202_2048 = +0.814036 -0.580814
0110100000_1011010101 // W0203_2048 = +0.812251 -0.583309
0110011111_1011010100 // W0204_2048 = +0.810457 -0.585798
0110011110_1011010011 // W0205_2048 = +0.808656 -0.588282
0110011101_1011010010 // W0206_2048 = +0.806848 -0.590760
0110011100_1011010000 // W0207_2048 = +0.805031 -0.593232
0110011011_1011001111 // W0208_2048 = +0.803208 -0.595699
0110011010_1011001110 // W0209_2048 = +0.801376 -0.598161
0110011001_1011001100 // W0210_2048 = +0.799537 -0.600616
0110011000_1011001011 // W0211_2048 = +0.797691 -0.603067
0110010111_1011001010 // W0212_2048 = +0.795837 -0.605511
0110010111_1011001001 // W0213_2048 = +0.793975 -0.607950
0110010110_1011000111 // W0214_2048 = +0.792107 -0.610383
0110010101_1011000110 // W0215_2048 = +0.790230 -0.612810
0110010100_1011000101 // W0216_2048 = +0.788346 -0.615232
0110010011_1011000100 // W0217_2048 = +0.786455 -0.617647
0110010010_1011000011 // W0218_2048 = +0.784557 -0.620057
0110010001_1011000001 // W0219_2048 = +0.782651 -0.622461
0110010000_1011000000 // W0220_2048 = +0.780737 -0.624859
0110001111_1010111111 // W0221_2048 = +0.778817 -0.627252
0110001110_1010111110 // W0222_2048 = +0.776888 -0.629638
0110001101_1010111100 // W0223_2048 = +0.774953 -0.632019
```

```
0110001100_1010111011  // W0224_2048 = +0.773010   -0.634393
0110001011_1010111010  // W0225_2048 = +0.771061   -0.636762
0110001010_1010111001  // W0226_2048 = +0.769103   -0.639124
0110001001_1010111000  // W0227_2048 = +0.767139   -0.641481
0110001000_1010110110  // W0228_2048 = +0.765167   -0.643832
0110000111_1010110101  // W0229_2048 = +0.763188   -0.646176
0110000110_1010110100  // W0230_2048 = +0.761202   -0.648514
0110000101_1010110011  // W0231_2048 = +0.759209   -0.650847
0110000100_1010110010  // W0232_2048 = +0.757209   -0.653173
0110000011_1010110000  // W0233_2048 = +0.755201   -0.655493
0110000010_1010101111  // W0234_2048 = +0.753187   -0.657807
0110000001_1010101110  // W0235_2048 = +0.751165   -0.660114
0110000000_1010101101  // W0236_2048 = +0.749136   -0.662416
0101111111_1010101100  // W0237_2048 = +0.747101   -0.664711
0101111101_1010101010  // W0238_2048 = +0.745058   -0.667000
0101111100_1010101001  // W0239_2048 = +0.743008   -0.669283
0101111011_1010101000  // W0240_2048 = +0.740951   -0.671559
0101111010_1010100111  // W0241_2048 = +0.738887   -0.673829
0101111001_1010100110  // W0242_2048 = +0.736817   -0.676093
0101111000_1010100101  // W0243_2048 = +0.734739   -0.678350
0101110111_1010100100  // W0244_2048 = +0.732654   -0.680601
0101110110_1010100010  // W0245_2048 = +0.730563   -0.682846
0101110101_1010100001  // W0246_2048 = +0.728464   -0.685084
0101110100_1010100000  // W0247_2048 = +0.726359   -0.687315
0101110011_1010011111  // W0248_2048 = +0.724247   -0.689541
0101110010_1010011110  // W0249_2048 = +0.722128   -0.691759
0101110001_1010011101  // W0250_2048 = +0.720003   -0.693971
0101110000_1010011100  // W0251_2048 = +0.717870   -0.696177
0101101110_1010011010  // W0252_2048 = +0.715731   -0.698376
0101101101_1010011001  // W0253_2048 = +0.713585   -0.700569
0101101100_1010011000  // W0254_2048 = +0.711432   -0.702755
0101101011_1010010111  // W0255_2048 = +0.709273   -0.704934
0101101010_1010010110  // W0256_2048 = +0.707107   -0.707107
0101101001_1010010101  // W0257_2048 = +0.704934   -0.709273
0101101000_1010010100  // W0258_2048 = +0.702755   -0.711432
0101100111_1010010011  // W0259_2048 = +0.700569   -0.713585
0101100110_1010010010  // W0260_2048 = +0.698376   -0.715731
0101100100_1010010000  // W0261_2048 = +0.696177   -0.717870
0101100011_1010001111  // W0262_2048 = +0.693971   -0.720003
0101100010_1010001110  // W0263_2048 = +0.691759   -0.722128
0101100001_1010001101  // W0264_2048 = +0.689541   -0.724247
0101100000_1010001100  // W0265_2048 = +0.687315   -0.726359
0101011111_1010001011  // W0266_2048 = +0.685084   -0.728464
0101011110_1010001010  // W0267_2048 = +0.682846   -0.730563
0101011100_1010001001  // W0268_2048 = +0.680601   -0.732654
0101011011_1010001000  // W0269_2048 = +0.678350   -0.734739
0101011010_1010000111  // W0270_2048 = +0.676093   -0.736817
0101011001_1010000110  // W0271_2048 = +0.673829   -0.738887
0101011000_1010000101  // W0272_2048 = +0.671559   -0.740951
0101010111_1010000100  // W0273_2048 = +0.669283   -0.743008
0101010110_1010000011  // W0274_2048 = +0.667000   -0.745058
0101010100_1010000001  // W0275_2048 = +0.664711   -0.747101
0101010011_1010000000  // W0276_2048 = +0.662416   -0.749136
0101010010_1001111111  // W0277_2048 = +0.660114   -0.751165
0101010001_1001111110  // W0278_2048 = +0.657807   -0.753187
0101010000_1001111101  // W0279_2048 = +0.655493   -0.755201
```

0101001110_1001111100	// W0280_2048 = +0.653173	-0.757209
0101001101_1001111011	// W0281_2048 = +0.650847	-0.759209
0101001100_1001111010	// W0282_2048 = +0.648514	-0.761202
0101001011_1001111001	// W0283_2048 = +0.646176	-0.763188
0101001010_1001111000	// W0284_2048 = +0.643832	-0.765167
0101001000_1001110111	// W0285_2048 = +0.641481	-0.767139
0101000111_1001110110	// W0286_2048 = +0.639124	-0.769103
0101000110_1001110101	// W0287_2048 = +0.636762	-0.771061
0101000101_1001110100	// W0288_2048 = +0.634393	-0.773010
0101000100_1001110011	// W0289_2048 = +0.632019	-0.774953
0101000010_1001110010	// W0290_2048 = +0.629638	-0.776888
0101000001_1001110001	// W0291_2048 = +0.627252	-0.778817
0101000000_1001110000	// W0292_2048 = +0.624859	-0.780737
0100111111_1001101111	// W0293_2048 = +0.622461	-0.782651
0100111101_1001101110	// W0294_2048 = +0.620057	-0.784557
0100111100_1001101101	// W0295_2048 = +0.617647	-0.786455
0100111011_1001101100	// W0296_2048 = +0.615232	-0.788346
0100111010_1001101011	// W0297_2048 = +0.612810	-0.790230
0100111001_1001101010	// W0298_2048 = +0.610383	-0.792107
0100110111_1001101001	// W0299_2048 = +0.607950	-0.793975
0100110110_1001101001	// W0300_2048 = +0.605511	-0.795837
0100110101_1001101000	// W0301_2048 = +0.603067	-0.797691
0100110100_1001100111	// W0302_2048 = +0.600616	-0.799537
0100110010_1001100110	// W0303_2048 = +0.598161	-0.801376
0100110001_1001100101	// W0304_2048 = +0.595699	-0.803208
0100110000_1001100100	// W0305_2048 = +0.593232	-0.805031
0100101110_1001100011	// W0306_2048 = +0.590760	-0.806848
0100101101_1001100010	// W0307_2048 = +0.588282	-0.808656
0100101100_1001100001	// W0308_2048 = +0.585798	-0.810457
0100101011_1001100000	// W0309_2048 = +0.583309	-0.812251
0100101001_1001011111	// W0310_2048 = +0.580814	-0.814036
0100101000_1001011110	// W0311_2048 = +0.578314	-0.815814
0100100111_1001011101	// W0312_2048 = +0.575808	-0.817585
0100100110_1001011100	// W0313_2048 = +0.573297	-0.819348
0100100100_1001011100	// W0314_2048 = +0.570781	-0.821103
0100100011_1001011011	// W0315_2048 = +0.568259	-0.822850
0100100010_1001011010	// W0316_2048 = +0.565732	-0.824589
0100100000_1001011001	// W0317_2048 = +0.563199	-0.826321
0100011111_1001011000	// W0318_2048 = +0.560662	-0.828045
0100011110_1001010111	// W0319_2048 = +0.558119	-0.829761
0100011100_1001010110	// W0320_2048 = +0.555570	-0.831470
0100011011_1001010101	// W0321_2048 = +0.553017	-0.833170
0100011010_1001010101	// W0322_2048 = +0.550458	-0.834863
0100011001_1001010100	// W0323_2048 = +0.547894	-0.836548
0100010111_1001010011	// W0324_2048 = +0.545325	-0.838225
0100010110_1001010010	// W0325_2048 = +0.542751	-0.839894
0100010101_1001010001	// W0326_2048 = +0.540171	-0.841555
0100010011_1001010000	// W0327_2048 = +0.537587	-0.843208
0100010010_1001001111	// W0328_2048 = +0.534998	-0.844854
0100010001_1001001111	// W0329_2048 = +0.532403	-0.846491
0100001111_1001001110	// W0330_2048 = +0.529804	-0.848120
0100001110_1001001101	// W0331_2048 = +0.527199	-0.849742
0100001101_1001001100	// W0332_2048 = +0.524590	-0.851355
0100001011_1001001011	// W0333_2048 = +0.521975	-0.852961
0100001010_1001001010	// W0334_2048 = +0.519356	-0.854558
0100001001_1001001010	// W0335_2048 = +0.516732	-0.856147

```
0100000111_1001001001  // W0336_2048 = +0.514103  -0.857729
0100000110_1001001000  // W0337_2048 = +0.511469  -0.859302
0100000101_1001000111  // W0338_2048 = +0.508830  -0.860867
0100000011_1001000110  // W0339_2048 = +0.506187  -0.862424
0100000010_1001000110  // W0340_2048 = +0.503538  -0.863973
0100000000_1001000101  // W0341_2048 = +0.500885  -0.865514
0011111111_1001000100  // W0342_2048 = +0.498228  -0.867046
0011111110_1001000011  // W0343_2048 = +0.495565  -0.868571
0011111100_1001000011  // W0344_2048 = +0.492898  -0.870087
0011111011_1001000010  // W0345_2048 = +0.490226  -0.871595
0011111010_1001000001  // W0346_2048 = +0.487550  -0.873095
0011111000_1001000000  // W0347_2048 = +0.484869  -0.874587
0011110111_1000111111  // W0348_2048 = +0.482184  -0.876070
0011110110_1000111111  // W0349_2048 = +0.479494  -0.877545
0011110100_1000111110  // W0350_2048 = +0.476799  -0.879012
0011110011_1000111101  // W0351_2048 = +0.474100  -0.880471
0011110001_1000111100  // W0352_2048 = +0.471397  -0.881921
0011110000_1000111100  // W0353_2048 = +0.468689  -0.883363
0011101111_1000111011  // W0354_2048 = +0.465976  -0.884797
0011101101_1000111010  // W0355_2048 = +0.463260  -0.886223
0011101100_1000111010  // W0356_2048 = +0.460539  -0.887640
0011101010_1000111001  // W0357_2048 = +0.457813  -0.889048
0011101001_1000111000  // W0358_2048 = +0.455084  -0.890449
0011101000_1000110111  // W0359_2048 = +0.452350  -0.891841
0011100110_1000110111  // W0360_2048 = +0.449611  -0.893224
0011100101_1000110110  // W0361_2048 = +0.446869  -0.894599
0011100011_1000110101  // W0362_2048 = +0.444122  -0.895966
0011100010_1000110101  // W0363_2048 = +0.441371  -0.897325
0011100001_1000110100  // W0364_2048 = +0.438616  -0.898674
0011011111_1000110011  // W0365_2048 = +0.435857  -0.900016
0011011110_1000110011  // W0366_2048 = +0.433094  -0.901349
0011011100_1000110010  // W0367_2048 = +0.430326  -0.902673
0011011011_1000110001  // W0368_2048 = +0.427555  -0.903989
0011011001_1000110000  // W0369_2048 = +0.424780  -0.905297
0011011000_1000110000  // W0370_2048 = +0.422000  -0.906596
0011010111_1000101111  // W0371_2048 = +0.419217  -0.907886
0011010101_1000101111  // W0372_2048 = +0.416430  -0.909168
0011010100_1000101110  // W0373_2048 = +0.413638  -0.910441
0011010010_1000101101  // W0374_2048 = +0.410843  -0.911706
0011010001_1000101101  // W0375_2048 = +0.408044  -0.912962
0011001111_1000101100  // W0376_2048 = +0.405241  -0.914210
0011001110_1000101011  // W0377_2048 = +0.402435  -0.915449
0011001101_1000101011  // W0378_2048 = +0.399624  -0.916679
0011001011_1000101010  // W0379_2048 = +0.396810  -0.917901
0011001010_1000101001  // W0380_2048 = +0.393992  -0.919114
0011001000_1000101001  // W0381_2048 = +0.391170  -0.920318
0011000111_1000101000  // W0382_2048 = +0.388345  -0.921514
0011000101_1000101000  // W0383_2048 = +0.385516  -0.922701
0011000100_1000100111  // W0384_2048 = +0.382683  -0.923880
0011000010_1000100110  // W0385_2048 = +0.379847  -0.925049
0011000001_1000100110  // W0386_2048 = +0.377007  -0.926210
0011000000_1000100101  // W0387_2048 = +0.374164  -0.927363
0010111110_1000100101  // W0388_2048 = +0.371317  -0.928506
0010111101_1000100100  // W0389_2048 = +0.368467  -0.929641
0010111011_1000100011  // W0390_2048 = +0.365613  -0.930767
0010111010_1000100011  // W0391_2048 = +0.362756  -0.931884
```

0010111000_1000100010	// W0392_2048 = +0.359895	-0.932993
0010110111_1000100010	// W0393_2048 = +0.357031	-0.934093
0010110101_1000100001	// W0394_2048 = +0.354164	-0.935184
0010110100_1000100001	// W0395_2048 = +0.351293	-0.936266
0010110010_1000100000	// W0396_2048 = +0.348419	-0.937339
0010110001_1000100000	// W0397_2048 = +0.345541	-0.938404
0010101111_1000011111	// W0398_2048 = +0.342661	-0.939459
0010101110_1000011110	// W0399_2048 = +0.339777	-0.940506
0010101100_1000011110	// W0400_2048 = +0.336890	-0.941544
0010101011_1000011101	// W0401_2048 = +0.334000	-0.942573
0010101010_1000011101	// W0402_2048 = +0.331106	-0.943593
0010101000_1000011100	// W0403_2048 = +0.328210	-0.944605
0010100111_1000011100	// W0404_2048 = +0.325310	-0.945607
0010100101_1000011011	// W0405_2048 = +0.322408	-0.946601
0010100100_1000011011	// W0406_2048 = +0.319502	-0.947586
0010100010_1000011010	// W0407_2048 = +0.316593	-0.948561
0010100001_1000011010	// W0408_2048 = +0.313682	-0.949528
0010011111_1000011001	// W0409_2048 = +0.310767	-0.950486
0010011110_1000011001	// W0410_2048 = +0.307850	-0.951435
0010011100_1000011000	// W0411_2048 = +0.304929	-0.952375
0010011011_1000011000	// W0412_2048 = +0.302006	-0.953306
0010011001_1000010111	// W0413_2048 = +0.299080	-0.954228
0010011000_1000010111	// W0414_2048 = +0.296151	-0.955141
0010010110_1000010111	// W0415_2048 = +0.293219	-0.956045
0010010101_1000010110	// W0416_2048 = +0.290285	-0.956940
0010010011_1000010110	// W0417_2048 = +0.287347	-0.957826
0010010010_1000010101	// W0418_2048 = +0.284408	-0.958703
0010010000_1000010101	// W0419_2048 = +0.281465	-0.959572
0010001111_1000010100	// W0420_2048 = +0.278520	-0.960431
0010001101_1000010100	// W0421_2048 = +0.275572	-0.961280
0010001100_1000010011	// W0422_2048 = +0.272621	-0.962121
0010001010_1000010011	// W0423_2048 = +0.269668	-0.962953
0010001001_1000010011	// W0424_2048 = +0.266713	-0.963776
0010000111_1000010010	// W0425_2048 = +0.263755	-0.964590
0010000110_1000010010	// W0426_2048 = +0.260794	-0.965394
0010000100_1000010001	// W0427_2048 = +0.257831	-0.966190
0010000010_1000010001	// W0428_2048 = +0.254866	-0.966976
0010000001_1000010001	// W0429_2048 = +0.251898	-0.967754
0001111111_1000010000	// W0430_2048 = +0.248928	-0.968522
0001111110_1000010000	// W0431_2048 = +0.245955	-0.969281
0001111100_1000001111	// W0432_2048 = +0.242980	-0.970031
0001111011_1000001111	// W0433_2048 = +0.240003	-0.970772
0001111001_1000001111	// W0434_2048 = +0.237024	-0.971504
0001111000_1000001110	// W0435_2048 = +0.234042	-0.972226
0001110110_1000001110	// W0436_2048 = +0.231058	-0.972940
0001110101_1000001101	// W0437_2048 = +0.228072	-0.973644
0001110011_1000001101	// W0438_2048 = +0.225084	-0.974339
0001110010_1000001101	// W0439_2048 = +0.222094	-0.975025
0001110000_1000001100	// W0440_2048 = +0.219101	-0.975702
0001101111_1000001100	// W0441_2048 = +0.216107	-0.976370
0001101101_1000001100	// W0442_2048 = +0.213110	-0.977028
0001101100_1000001011	// W0443_2048 = +0.210112	-0.977677
0001101010_1000001011	// W0444_2048 = +0.207111	-0.978317
0001101001_1000001011	// W0445_2048 = +0.204109	-0.978948
0001100111_1000001010	// W0446_2048 = +0.201105	-0.979570
0001100101_1000001010	// W0447_2048 = +0.198098	-0.980182

```
0001100100_1000001010  // W0448_2048 = +0.195090   -0.980785
0001100010_1000001010  // W0449_2048 = +0.192080   -0.981379
0001100001_1000001001  // W0450_2048 = +0.189069   -0.981964
0001011111_1000001001  // W0451_2048 = +0.186055   -0.982539
0001011110_1000001001  // W0452_2048 = +0.183040   -0.983105
0001011100_1000001000  // W0453_2048 = +0.180023   -0.983662
0001011011_1000001000  // W0454_2048 = +0.177004   -0.984210
0001011001_1000001000  // W0455_2048 = +0.173984   -0.984749
0001011000_1000001000  // W0456_2048 = +0.170962   -0.985278
0001010110_1000000111  // W0457_2048 = +0.167938   -0.985798
0001010100_1000000111  // W0458_2048 = +0.164913   -0.986308
0001010011_1000000111  // W0459_2048 = +0.161886   -0.986809
0001010001_1000000111  // W0460_2048 = +0.158858   -0.987301
0001010000_1000000110  // W0461_2048 = +0.155828   -0.987784
0001001110_1000000110  // W0462_2048 = +0.152797   -0.988258
0001001101_1000000110  // W0463_2048 = +0.149765   -0.988722
0001001011_1000000110  // W0464_2048 = +0.146730   -0.989177
0001001010_1000000101  // W0465_2048 = +0.143695   -0.989622
0001001000_1000000101  // W0466_2048 = +0.140658   -0.990058
0001000110_1000000101  // W0467_2048 = +0.137620   -0.990485
0001000101_1000000101  // W0468_2048 = +0.134581   -0.990903
0001000011_1000000100  // W0469_2048 = +0.131540   -0.991311
0001000010_1000000100  // W0470_2048 = +0.128498   -0.991710
0001000000_1000000100  // W0471_2048 = +0.125455   -0.992099
0000111111_1000000100  // W0472_2048 = +0.122411   -0.992480
0000111101_1000000100  // W0473_2048 = +0.119365   -0.992850
0000111100_1000000011  // W0474_2048 = +0.116319   -0.993212
0000111010_1000000011  // W0475_2048 = +0.113271   -0.993564
0000111000_1000000011  // W0476_2048 = +0.110222   -0.993907
0000110111_1000000011  // W0477_2048 = +0.107172   -0.994240
0000110101_1000000011  // W0478_2048 = +0.104122   -0.994565
0000110100_1000000011  // W0479_2048 = +0.101070   -0.994879
0000110010_1000000010  // W0480_2048 = +0.098017   -0.995185
0000110001_1000000010  // W0481_2048 = +0.094963   -0.995481
0000101111_1000000010  // W0482_2048 = +0.091909   -0.995767
0000101101_1000000010  // W0483_2048 = +0.088854   -0.996045
0000101100_1000000010  // W0484_2048 = +0.085797   -0.996313
0000101010_1000000010  // W0485_2048 = +0.082740   -0.996571
0000101001_1000000010  // W0486_2048 = +0.079682   -0.996820
0000100111_1000000010  // W0487_2048 = +0.076624   -0.997060
0000100110_1000000001  // W0488_2048 = +0.073565   -0.997290
0000100100_1000000001  // W0489_2048 = +0.070505   -0.997511
0000100011_1000000001  // W0490_2048 = +0.067444   -0.997723
0000100001_1000000001  // W0491_2048 = +0.064383   -0.997925
0000011111_1000000001  // W0492_2048 = +0.061321   -0.998118
0000011110_1000000001  // W0493_2048 = +0.058258   -0.998302
0000011100_1000000001  // W0494_2048 = +0.055195   -0.998476
0000011011_1000000001  // W0495_2048 = +0.052132   -0.998640
0000011001_1000000001  // W0496_2048 = +0.049068   -0.998795
0000011000_1000000001  // W0497_2048 = +0.046003   -0.998941
0000010110_1000000000  // W0498_2048 = +0.042938   -0.999078
0000010100_1000000000  // W0499_2048 = +0.039873   -0.999205
0000010011_1000000000  // W0500_2048 = +0.036807   -0.999322
0000010001_1000000000  // W0501_2048 = +0.033741   -0.999431
0000010000_1000000000  // W0502_2048 = +0.030675   -0.999529
0000001110_1000000000  // W0503_2048 = +0.027608   -0.999619
```

```
0000001101_1000000000  // W0504_2048 = +0.024541   -0.999699
0000001011_1000000000  // W0505_2048 = +0.021474   -0.999769
0000001001_1000000000  // W0506_2048 = +0.018407   -0.999831
0000001000_1000000000  // W0507_2048 = +0.015339   -0.999882
0000000110_1000000000  // W0508_2048 = +0.012272   -0.999925
0000000101_1000000000  // W0509_2048 = +0.009204   -0.999958
0000000011_1000000000  // W0510_2048 = +0.006136   -0.999981
0000000010_1000000000  // W0511_2048 = +0.003068   -0.999995
0000000000_1000000000  // W0512_2048 = +0.000000   -1.000000
1111111110_1000000000  // W0513_2048 = -0.003068   -0.999995
1111111101_1000000000  // W0514_2048 = -0.006136   -0.999981
1111111010_1000000000  // W0516_2048 = -0.012272   -0.999925
1111110111_1000000000  // W0518_2048 = -0.018407   -0.999831
1111110101_1000000000  // W0519_2048 = -0.021474   -0.999769
1111110011_1000000000  // W0520_2048 = -0.024541   -0.999699
1111110000_1000000000  // W0522_2048 = -0.030675   -0.999529
1111101101_1000000000  // W0524_2048 = -0.036807   -0.999322
1111101100_1000000000  // W0525_2048 = -0.039873   -0.999205
1111101010_1000000000  // W0526_2048 = -0.042938   -0.999078
1111100111_1000000001  // W0528_2048 = -0.049068   -0.998795
1111100100_1000000001  // W0530_2048 = -0.055195   -0.998476
1111100010_1000000001  // W0531_2048 = -0.058258   -0.998302
1111100001_1000000001  // W0532_2048 = -0.061321   -0.998118
1111011101_1000000001  // W0534_2048 = -0.067444   -0.997723
1111011010_1000000001  // W0536_2048 = -0.073565   -0.997290
1111011001_1000000010  // W0537_2048 = -0.076624   -0.997060
1111010111_1000000010  // W0538_2048 = -0.079682   -0.996820
1111010100_1000000010  // W0540_2048 = -0.085797   -0.996313
1111010001_1000000010  // W0542_2048 = -0.091909   -0.995767
1111001111_1000000010  // W0543_2048 = -0.094963   -0.995481
1111001110_1000000010  // W0544_2048 = -0.098017   -0.995185
1111001011_1000000011  // W0546_2048 = -0.104122   -0.994565
1111001000_1000000011  // W0548_2048 = -0.110222   -0.993907
1111000110_1000000011  // W0549_2048 = -0.113271   -0.993564
1111000100_1000000011  // W0550_2048 = -0.116319   -0.993212
1111000001_1000000100  // W0552_2048 = -0.122411   -0.992480
1110111110_1000000100  // W0554_2048 = -0.128498   -0.991710
1110111101_1000000100  // W0555_2048 = -0.131540   -0.991311
1110111011_1000000101  // W0556_2048 = -0.134581   -0.990903
1110111000_1000000101  // W0558_2048 = -0.140658   -0.990058
1110110101_1000000110  // W0560_2048 = -0.146730   -0.989177
1110110011_1000000110  // W0561_2048 = -0.149765   -0.988722
1110110010_1000000110  // W0562_2048 = -0.152797   -0.988258
1110101111_1000000111  // W0564_2048 = -0.158858   -0.987301
1110101100_1000000111  // W0566_2048 = -0.164913   -0.986308
1110101010_1000000111  // W0567_2048 = -0.167938   -0.985798
1110101000_1000001000  // W0568_2048 = -0.170962   -0.985278
1110100101_1000001000  // W0570_2048 = -0.177004   -0.984210
1110100010_1000001001  // W0572_2048 = -0.183040   -0.983105
1110100001_1000001001  // W0573_2048 = -0.186055   -0.982539
1110011111_1000001001  // W0574_2048 = -0.189069   -0.981964
1110011100_1000001010  // W0576_2048 = -0.195090   -0.980785
1110011001_1000001010  // W0578_2048 = -0.201105   -0.979570
1110010111_1000001011  // W0579_2048 = -0.204109   -0.978948
1110010110_1000001011  // W0580_2048 = -0.207111   -0.978317
1110010011_1000001100  // W0582_2048 = -0.213110   -0.977028
```

1110010000_1000001100	// W0584_2048 = -0.219101	-0.975702
1110001110_1000001101	// W0585_2048 = -0.222094	-0.975025
1110001101_1000001101	// W0586_2048 = -0.225084	-0.974339
1110001010_1000001110	// W0588_2048 = -0.231058	-0.972940
1110000111_1000001111	// W0590_2048 = -0.237024	-0.971504
1110000101_1000001111	// W0591_2048 = -0.240003	-0.970772
1110000100_1000001111	// W0592_2048 = -0.242980	-0.970031
1110000001_1000010000	// W0594_2048 = -0.248928	-0.968522
1101111110_1000010001	// W0596_2048 = -0.254866	-0.966976
1101111100_1000010001	// W0597_2048 = -0.257831	-0.966190
1101111010_1000010010	// W0598_2048 = -0.260794	-0.965394
1101110111_1000010011	// W0600_2048 = -0.266713	-0.963776
1101110100_1000010011	// W0602_2048 = -0.272621	-0.962121
1101110011_1000010100	// W0603_2048 = -0.275572	-0.961280
1101110001_1000010100	// W0604_2048 = -0.278520	-0.960431
1101101110_1000010101	// W0606_2048 = -0.284408	-0.958703
1101101011_1000010110	// W0608_2048 = -0.290285	-0.956940
1101101010_1000010111	// W0609_2048 = -0.293219	-0.956045
1101101000_1000010111	// W0610_2048 = -0.296151	-0.955141
1101100101_1000011000	// W0612_2048 = -0.302006	-0.953306
1101100010_1000011001	// W0614_2048 = -0.307850	-0.951435
1101100001_1000011001	// W0615_2048 = -0.310767	-0.950486
1101011111_1000011010	// W0616_2048 = -0.313682	-0.949528
1101011100_1000011011	// W0618_2048 = -0.319502	-0.947586
1101011001_1000011100	// W0620_2048 = -0.325310	-0.945607
1101011000_1000011100	// W0621_2048 = -0.328210	-0.944605
1101010110_1000011101	// W0622_2048 = -0.331106	-0.943593
1101010100_1000011110	// W0624_2048 = -0.336890	-0.941544
1101010001_1000011111	// W0626_2048 = -0.342661	-0.939459
1101001111_1000100000	// W0627_2048 = -0.345541	-0.938404
1101001110_1000100000	// W0628_2048 = -0.348419	-0.937339
1101001011_1000100001	// W0630_2048 = -0.354164	-0.935184
1101001000_1000100010	// W0632_2048 = -0.359895	-0.932993
1101000110_1000100011	// W0633_2048 = -0.362756	-0.931884
1101000101_1000100011	// W0634_2048 = -0.365613	-0.930767
1101000010_1000100101	// W0636_2048 = -0.371317	-0.928506
1100111111_1000100110	// W0638_2048 = -0.377007	-0.926210
1100111110_1000100110	// W0639_2048 = -0.379847	-0.925049
1100111100_1000100111	// W0640_2048 = -0.382683	-0.923880
1100111001_1000101000	// W0642_2048 = -0.388345	-0.921514
1100110110_1000101001	// W0644_2048 = -0.393992	-0.919114
1100110101_1000101010	// W0645_2048 = -0.396810	-0.917901
1100110011_1000101011	// W0646_2048 = -0.399624	-0.916679
1100110001_1000101100	// W0648_2048 = -0.405241	-0.914210
1100101110_1000101101	// W0650_2048 = -0.410843	-0.911706
1100101100_1000101110	// W0651_2048 = -0.413638	-0.910441
1100101011_1000101111	// W0652_2048 = -0.416430	-0.909168
1100101000_1000110000	// W0654_2048 = -0.422000	-0.906596
1100100101_1000110001	// W0656_2048 = -0.427555	-0.903989
1100100100_1000110010	// W0657_2048 = -0.430326	-0.902673
1100100010_1000110011	// W0658_2048 = -0.433094	-0.901349
1100011111_1000110100	// W0660_2048 = -0.438616	-0.898674
1100011101_1000110101	// W0662_2048 = -0.444122	-0.895966
1100011011_1000110110	// W0663_2048 = -0.446869	-0.894599
1100011010_1000110111	// W0664_2048 = -0.449611	-0.893224
1100010111_1000111000	// W0666_2048 = -0.455084	-0.890449

1100010100_1000111010	// W0668_2048 = -0.460539	-0.887640
1100010011_1000111010	// W0669_2048 = -0.463260	-0.886223
1100010001_1000111011	// W0670_2048 = -0.465976	-0.884797
1100001111_1000111100	// W0672_2048 = -0.471397	-0.881921
1100001100_1000111110	// W0674_2048 = -0.476799	-0.879012
1100001010_1000111111	// W0675_2048 = -0.479494	-0.877545
1100001001_1000111111	// W0676_2048 = -0.482184	-0.876070
1100000110_1001000001	// W0678_2048 = -0.487550	-0.873095
1100000100_1001000011	// W0680_2048 = -0.492898	-0.870087
1100000010_1001000011	// W0681_2048 = -0.495565	-0.868571
1100000001_1001000100	// W0682_2048 = -0.498228	-0.867046
1011111110_1001000110	// W0684_2048 = -0.503538	-0.863973
1011111011_1001000111	// W0686_2048 = -0.508830	-0.860867
1011111010_1001001000	// W0687_2048 = -0.511469	-0.859302
1011111001_1001001001	// W0688_2048 = -0.514103	-0.857729
1011110110_1001001010	// W0690_2048 = -0.519356	-0.854558
1011110011_1001001100	// W0692_2048 = -0.524590	-0.851355
1011110010_1001001101	// W0693_2048 = -0.527199	-0.849742
1011110001_1001001110	// W0694_2048 = -0.529804	-0.848120
1011101110_1001001111	// W0696_2048 = -0.534998	-0.844854
1011101011_1001010001	// W0698_2048 = -0.540171	-0.841555
1011101010_1001010010	// W0699_2048 = -0.542751	-0.839894
1011101001_1001010011	// W0700_2048 = -0.545325	-0.838225
1011100110_1001010101	// W0702_2048 = -0.550458	-0.834863
1011100100_1001010110	// W0704_2048 = -0.555570	-0.831470
1011100010_1001010111	// W0705_2048 = -0.558119	-0.829761
1011100001_1001011000	// W0706_2048 = -0.560662	-0.828045
1011011110_1001011010	// W0708_2048 = -0.565732	-0.824589
1011011100_1001011100	// W0710_2048 = -0.570781	-0.821103
1011011010_1001011100	// W0711_2048 = -0.573297	-0.819348
1011011001_1001011101	// W0712_2048 = -0.575808	-0.817585
1011010111_1001011111	// W0714_2048 = -0.580814	-0.814036
1011010100_1001100001	// W0716_2048 = -0.585798	-0.810457
1011010011_1001100010	// W0717_2048 = -0.588282	-0.808656
1011010010_1001100011	// W0718_2048 = -0.590760	-0.806848
1011001111_1001100101	// W0720_2048 = -0.595699	-0.803208
1011001100_1001100111	// W0722_2048 = -0.600616	-0.799537
1011001011_1001101000	// W0723_2048 = -0.603067	-0.797691
1011001010_1001101001	// W0724_2048 = -0.605511	-0.795837
1011000111_1001101010	// W0726_2048 = -0.610383	-0.792107
1011000101_1001101100	// W0728_2048 = -0.615232	-0.788346
1011000100_1001101101	// W0729_2048 = -0.617647	-0.786455
1011000011_1001101110	// W0730_2048 = -0.620057	-0.784557
1011000000_1001110000	// W0732_2048 = -0.624859	-0.780737
1010111110_1001110010	// W0734_2048 = -0.629638	-0.776888
1010111100_1001110011	// W0735_2048 = -0.632019	-0.774953
1010111011_1001110100	// W0736_2048 = -0.634393	-0.773010
1010111001_1001110110	// W0738_2048 = -0.639124	-0.769103
1010110110_1001111000	// W0740_2048 = -0.643832	-0.765167
1010110101_1001111001	// W0741_2048 = -0.646176	-0.763188
1010110100_1001111010	// W0742_2048 = -0.648514	-0.761202
1010110010_1001111100	// W0744_2048 = -0.653173	-0.757209
1010101111_1001111110	// W0746_2048 = -0.657807	-0.753187
1010101110_1001111111	// W0747_2048 = -0.660114	-0.751165
1010101101_1010000000	// W0748_2048 = -0.662416	-0.749136
1010101010_1010000011	// W0750_2048 = -0.667000	-0.745058

1010101000_1010000101	// W0752_2048 = -0.671559	-0.740951
1010100111_1010000110	// W0753_2048 = -0.673829	-0.738887
1010100110_1010000111	// W0754_2048 = -0.676093	-0.736817
1010100100_1010001001	// W0756_2048 = -0.680601	-0.732654
1010100001_1010001011	// W0758_2048 = -0.685084	-0.728464
1010100000_1010001100	// W0759_2048 = -0.687315	-0.726359
1010011111_1010001101	// W0760_2048 = -0.689541	-0.724247
1010011101_1010001111	// W0762_2048 = -0.693971	-0.720003
1010011010_1010010010	// W0764_2048 = -0.698376	-0.715731
1010011001_1010010011	// W0765_2048 = -0.700569	-0.713585
1010011000_1010010100	// W0766_2048 = -0.702755	-0.711432
1010010110_1010010110	// W0768_2048 = -0.707107	-0.707107
1010010100_1010011000	// W0770_2048 = -0.711432	-0.702755
1010010011_1010011001	// W0771_2048 = -0.713585	-0.700569
1010010010_1010011010	// W0772_2048 = -0.715731	-0.698376
1010001111_1010011101	// W0774_2048 = -0.720003	-0.693971
1010001101_1010011111	// W0776_2048 = -0.724247	-0.689541
1010001100_1010100000	// W0777_2048 = -0.726359	-0.687315
1010001011_1010100001	// W0778_2048 = -0.728464	-0.685084
1010001001_1010100100	// W0780_2048 = -0.732654	-0.680601
1010000111_1010100110	// W0782_2048 = -0.736817	-0.676093
1010000110_1010100111	// W0783_2048 = -0.738887	-0.673829
1010000101_1010101000	// W0784_2048 = -0.740951	-0.671559
1010000011_1010101010	// W0786_2048 = -0.745058	-0.667000
1010000000_1010101101	// W0788_2048 = -0.749136	-0.662416
1001111111_1010101110	// W0789_2048 = -0.751165	-0.660114
1001111110_1010101111	// W0790_2048 = -0.753187	-0.657807
1001111100_1010110010	// W0792_2048 = -0.757209	-0.653173
1001111010_1010110100	// W0794_2048 = -0.761202	-0.648514
1001111001_1010110101	// W0795_2048 = -0.763188	-0.646176
1001111000_1010110110	// W0796_2048 = -0.765167	-0.643832
1001110110_1010111001	// W0798_2048 = -0.769103	-0.639124
1001110100_1010111011	// W0800_2048 = -0.773010	-0.634393
1001110011_1010111100	// W0801_2048 = -0.774953	-0.632019
1001110010_1010111110	// W0802_2048 = -0.776888	-0.629638
1001110000_1011000000	// W0804_2048 = -0.780737	-0.624859
1001101110_1011000011	// W0806_2048 = -0.784557	-0.620057
1001101101_1011000100	// W0807_2048 = -0.786455	-0.617647
1001101100_1011000101	// W0808_2048 = -0.788346	-0.615232
1001101010_1011000111	// W0810_2048 = -0.792107	-0.610383
1001101001_1011001010	// W0812_2048 = -0.795837	-0.605511
1001101000_1011001011	// W0813_2048 = -0.797691	-0.603067
1001100111_1011001100	// W0814_2048 = -0.799537	-0.600616
1001100101_1011001111	// W0816_2048 = -0.803208	-0.595699
1001100011_1011010010	// W0818_2048 = -0.806848	-0.590760
1001100010_1011010011	// W0819_2048 = -0.808656	-0.588282
1001100001_1011010100	// W0820_2048 = -0.810457	-0.585798
1001011111_1011010111	// W0822_2048 = -0.814036	-0.580814
1001011101_1011011001	// W0824_2048 = -0.817585	-0.575808
1001011100_1011011010	// W0825_2048 = -0.819348	-0.573297
1001011100_1011011100	// W0826_2048 = -0.821103	-0.570781
1001011010_1011011110	// W0828_2048 = -0.824589	-0.565732
1001011000_1011100001	// W0830_2048 = -0.828045	-0.560662
1001010111_1011100010	// W0831_2048 = -0.829761	-0.558119
1001010110_1011100100	// W0832_2048 = -0.831470	-0.555570
1001010101_1011100110	// W0834_2048 = -0.834863	-0.550458

1001010011_1011101001	// W0836_2048 = -0.838225	-0.545325
1001010010_1011101010	// W0837_2048 = -0.839894	-0.542751
1001010001_1011101011	// W0838_2048 = -0.841555	-0.540171
1001001111_1011101110	// W0840_2048 = -0.844854	-0.534998
1001001110_1011110001	// W0842_2048 = -0.848120	-0.529804
1001001101_1011110010	// W0843_2048 = -0.849742	-0.527199
1001001100_1011110011	// W0844_2048 = -0.851355	-0.524590
1001001010_1011110110	// W0846_2048 = -0.854558	-0.519356
1001001001_1011111001	// W0848_2048 = -0.857729	-0.514103
1001001000_1011111010	// W0849_2048 = -0.859302	-0.511469
1001000111_1011111011	// W0850_2048 = -0.860867	-0.508830
1001000110_1011111110	// W0852_2048 = -0.863973	-0.503538
1001000100_1100000001	// W0854_2048 = -0.867046	-0.498228
1001000011_1100000010	// W0855_2048 = -0.868571	-0.495565
1001000011_1100000100	// W0856_2048 = -0.870087	-0.492898
1001000001_1100000110	// W0858_2048 = -0.873095	-0.487550
1000111111_1100001001	// W0860_2048 = -0.876070	-0.482184
1000111111_1100001010	// W0861_2048 = -0.877545	-0.479494
1000111110_1100001100	// W0862_2048 = -0.879012	-0.476799
1000111100_1100001111	// W0864_2048 = -0.881921	-0.471397
1000111011_1100010001	// W0866_2048 = -0.884797	-0.465976
1000111010_1100010011	// W0867_2048 = -0.886223	-0.463260
1000111010_1100010100	// W0868_2048 = -0.887640	-0.460539
1000111000_1100010111	// W0870_2048 = -0.890449	-0.455084
1000110111_1100011010	// W0872_2048 = -0.893224	-0.449611
1000110110_1100011011	// W0873_2048 = -0.894599	-0.446869
1000110101_1100011101	// W0874_2048 = -0.895966	-0.444122
1000110100_1100011111	// W0876_2048 = -0.898674	-0.438616
1000110011_1100100010	// W0878_2048 = -0.901349	-0.433094
1000110010_1100100100	// W0879_2048 = -0.902673	-0.430326
1000110001_1100100101	// W0880_2048 = -0.903989	-0.427555
1000110000_1100101000	// W0882_2048 = -0.906596	-0.422000
1000101111_1100101011	// W0884_2048 = -0.909168	-0.416430
1000101110_1100101100	// W0885_2048 = -0.910441	-0.413638
1000101101_1100101110	// W0886_2048 = -0.911706	-0.410843
1000101100_1100110001	// W0888_2048 = -0.914210	-0.405241
1000101011_1100110011	// W0890_2048 = -0.916679	-0.399624
1000101010_1100110101	// W0891_2048 = -0.917901	-0.396810
1000101001_1100110110	// W0892_2048 = -0.919114	-0.393992
1000101000_1100111001	// W0894_2048 = -0.921514	-0.388345
1000100111_1100111100	// W0896_2048 = -0.923880	-0.382683
1000100110_1100111110	// W0897_2048 = -0.925049	-0.379847
1000100110_1100111111	// W0898_2048 = -0.926210	-0.377007
1000100101_1101000010	// W0900_2048 = -0.928506	-0.371317
1000100011_1101000101	// W0902_2048 = -0.930767	-0.365613
1000100011_1101000110	// W0903_2048 = -0.931884	-0.362756
1000100010_1101001000	// W0904_2048 = -0.932993	-0.359895
1000100001_1101001011	// W0906_2048 = -0.935184	-0.354164
1000100000_1101001110	// W0908_2048 = -0.937339	-0.348419
1000100000_1101001111	// W0909_2048 = -0.938404	-0.345541
1000011111_1101010001	// W0910_2048 = -0.939459	-0.342661
1000011110_1101010100	// W0912_2048 = -0.941544	-0.336890
1000011101_1101010110	// W0914_2048 = -0.943593	-0.331106
1000011100_1101011000	// W0915_2048 = -0.944605	-0.328210
1000011100_1101011001	// W0916_2048 = -0.945607	-0.325310
1000011011_1101011100	// W0918_2048 = -0.947586	-0.319502

1000011010_1101011111	// W0920_2048 = -0.949528	-0.313682
1000011001_1101100001	// W0921_2048 = -0.950486	-0.310767
1000011001_1101100010	// W0922_2048 = -0.951435	-0.307850
1000011000_1101100101	// W0924_2048 = -0.953306	-0.302006
1000010111_1101101000	// W0926_2048 = -0.955141	-0.296151
1000010111_1101101010	// W0927_2048 = -0.956045	-0.293219
1000010110_1101101011	// W0928_2048 = -0.956940	-0.290285
1000010101_1101101110	// W0930_2048 = -0.958703	-0.284408
1000010100_1101110001	// W0932_2048 = -0.960431	-0.278520
1000010100_1101110011	// W0933_2048 = -0.961280	-0.275572
1000010011_1101110100	// W0934_2048 = -0.962121	-0.272621
1000010011_1101110111	// W0936_2048 = -0.963776	-0.266713
1000010010_1101111010	// W0938_2048 = -0.965394	-0.260794
1000010001_1101111100	// W0939_2048 = -0.966190	-0.257831
1000010001_1101111110	// W0940_2048 = -0.966976	-0.254866
1000010000_1110000001	// W0942_2048 = -0.968522	-0.248928
1000001111_1110000100	// W0944_2048 = -0.970031	-0.242980
1000001111_1110000101	// W0945_2048 = -0.970772	-0.240003
1000001111_1110000111	// W0946_2048 = -0.971504	-0.237024
1000001110_1110001010	// W0948_2048 = -0.972940	-0.231058
1000001101_1110001101	// W0950_2048 = -0.974339	-0.225084
1000001101_1110001110	// W0951_2048 = -0.975025	-0.222094
1000001100_1110010000	// W0952_2048 = -0.975702	-0.219101
1000001100_1110010011	// W0954_2048 = -0.977028	-0.213110
1000001011_1110010110	// W0956_2048 = -0.978317	-0.207111
1000001011_1110010111	// W0957_2048 = -0.978948	-0.204109
1000001010_1110011001	// W0958_2048 = -0.979570	-0.201105
1000001010_1110011100	// W0960_2048 = -0.980785	-0.195090
1000001001_1110011111	// W0962_2048 = -0.981964	-0.189069
1000001001_1110100001	// W0963_2048 = -0.982539	-0.186055
1000001001_1110100010	// W0964_2048 = -0.983105	-0.183040
1000001000_1110100101	// W0966_2048 = -0.984210	-0.177004
1000001000_1110101000	// W0968_2048 = -0.985278	-0.170962
1000000111_1110101010	// W0969_2048 = -0.985798	-0.167938
1000000111_1110101100	// W0970_2048 = -0.986308	-0.164913
1000000111_1110101111	// W0972_2048 = -0.987301	-0.158858
1000000110_1110110010	// W0974_2048 = -0.988258	-0.152797
1000000110_1110110011	// W0975_2048 = -0.988722	-0.149765
1000000110_1110110101	// W0976_2048 = -0.989177	-0.146730
1000000101_1110111000	// W0978_2048 = -0.990058	-0.140658
1000000101_1110111011	// W0980_2048 = -0.990903	-0.134581
1000000100_1110111101	// W0981_2048 = -0.991311	-0.131540
1000000100_1110111110	// W0982_2048 = -0.991710	-0.128498
1000000100_1111000001	// W0984_2048 = -0.992480	-0.122411
1000000011_1111000100	// W0986_2048 = -0.993212	-0.116319
1000000011_1111000110	// W0987_2048 = -0.993564	-0.113271
1000000011_1111001000	// W0988_2048 = -0.993907	-0.110222
1000000011_1111001011	// W0990_2048 = -0.994565	-0.104122
1000000010_1111001110	// W0992_2048 = -0.995185	-0.098017
1000000010_1111001111	// W0993_2048 = -0.995481	-0.094963
1000000010_1111010001	// W0994_2048 = -0.995767	-0.091909
1000000010_1111010100	// W0996_2048 = -0.996313	-0.085797
1000000010_1111010111	// W0998_2048 = -0.996820	-0.079682
1000000010_1111011001	// W0999_2048 = -0.997060	-0.076624
1000000001_1111011010	// W1000_2048 = -0.997290	-0.073565
1000000001_1111011101	// W1002_2048 = -0.997723	-0.067444

1000000001_1111100001	// W1004_2048 = -0.998118	-0.061321
1000000001_1111100010	// W1005_2048 = -0.998302	-0.058258
1000000001_1111100100	// W1006_2048 = -0.998476	-0.055195
1000000001_1111100111	// W1008_2048 = -0.998795	-0.049068
1000000000_1111101010	// W1010_2048 = -0.999078	-0.042938
1000000000_1111101100	// W1011_2048 = -0.999205	-0.039873
1000000000_1111101101	// W1012_2048 = -0.999322	-0.036807
1000000000_1111110000	// W1014_2048 = -0.999529	-0.030675
1000000000_1111110011	// W1016_2048 = -0.999699	-0.024541
1000000000_1111110101	// W1017_2048 = -0.999769	-0.021474
1000000000_1111110111	// W1018_2048 = -0.999831	-0.018407
1000000000_1111111010	// W1020_2048 = -0.999925	-0.012272
1000000000_1111111101	// W1022_2048 = -0.999981	-0.006136
1000000000_1111111110	// W1023_2048 = -0.999995	-0.003068
1000000000_0000000011	// W1026_2048 = -0.999981	+0.006136
1000000000_0000001000	// W1029_2048 = -0.999882	+0.015339
1000000000_0000001101	// W1032_2048 = -0.999699	+0.024541
1000000000_0000010001	// W1035_2048 = -0.999431	+0.033741
1000000000_0000010110	// W1038_2048 = -0.999078	+0.042938
1000000001_0000011011	// W1041_2048 = -0.998640	+0.052132
1000000001_0000011111	// W1044_2048 = -0.998118	+0.061321
1000000001_0000100100	// W1047_2048 = -0.997511	+0.070505
1000000010_0000101001	// W1050_2048 = -0.996820	+0.079682
1000000010_0000101101	// W1053_2048 = -0.996045	+0.088854
1000000010_0000110010	// W1056_2048 = -0.995185	+0.098017
1000000011_0000110111	// W1059_2048 = -0.994240	+0.107172
1000000011_0000111100	// W1062_2048 = -0.993212	+0.116319
1000000100_0001000000	// W1065_2048 = -0.992099	+0.125455
1000000101_0001000101	// W1068_2048 = -0.990903	+0.134581
1000000101_0001001010	// W1071_2048 = -0.989622	+0.143695
1000000110_0001001110	// W1074_2048 = -0.988258	+0.152797
1000000111_0001010011	// W1077_2048 = -0.986809	+0.161886
1000001000_0001011000	// W1080_2048 = -0.985278	+0.170962
1000001000_0001011100	// W1083_2048 = -0.983662	+0.180023
1000001001_0001100001	// W1086_2048 = -0.981964	+0.189069
1000001010_0001100101	// W1089_2048 = -0.980182	+0.198098
1000001011_0001101010	// W1092_2048 = -0.978317	+0.207111
1000001100_0001101111	// W1095_2048 = -0.976370	+0.216107
1000001101_0001110011	// W1098_2048 = -0.974339	+0.225084
1000001110_0001111000	// W1101_2048 = -0.972226	+0.234042
1000001111_0001111100	// W1104_2048 = -0.970031	+0.242980
1000010001_0010000001	// W1107_2048 = -0.967754	+0.251898
1000010010_0010000110	// W1110_2048 = -0.965394	+0.260794
1000010011_0010001010	// W1113_2048 = -0.962953	+0.269668
1000010100_0010001111	// W1116_2048 = -0.960431	+0.278520
1000010110_0010010011	// W1119_2048 = -0.957826	+0.287347
1000010111_0010011000	// W1122_2048 = -0.955141	+0.296151
1000011000_0010011100	// W1125_2048 = -0.952375	+0.304929
1000011010_0010100001	// W1128_2048 = -0.949528	+0.313682
1000011011_0010100101	// W1131_2048 = -0.946601	+0.322408
1000011101_0010101010	// W1134_2048 = -0.943593	+0.331106
1000011110_0010101110	// W1137_2048 = -0.940506	+0.339777
1000100000_0010110010	// W1140_2048 = -0.937339	+0.348419
1000100010_0010110111	// W1143_2048 = -0.934093	+0.357031
1000100011_0010111011	// W1146_2048 = -0.930767	+0.365613
1000100101_0011000000	// W1149_2048 = -0.927363	+0.374164

```
1000100111_0011000100  // W1152_2048 = -0.923880  +0.382683
1000101001_0011001000  // W1155_2048 = -0.920318  +0.391170
1000101011_0011001101  // W1158_2048 = -0.916679  +0.399624
1000101101_0011010001  // W1161_2048 = -0.912962  +0.408044
1000101111_0011010101  // W1164_2048 = -0.909168  +0.416430
1000110000_0011011001  // W1167_2048 = -0.905297  +0.424780
1000110011_0011011110  // W1170_2048 = -0.901349  +0.433094
1000110101_0011100010  // W1173_2048 = -0.897325  +0.441371
1000110111_0011100110  // W1176_2048 = -0.893224  +0.449611
1000111001_0011101010  // W1179_2048 = -0.889048  +0.457813
1000111011_0011101111  // W1182_2048 = -0.884797  +0.465976
1000111101_0011110011  // W1185_2048 = -0.880471  +0.474100
1000111111_0011110111  // W1188_2048 = -0.876070  +0.482184
1001000010_0011111011  // W1191_2048 = -0.871595  +0.490226
1001000100_0011111111  // W1194_2048 = -0.867046  +0.498228
1001000110_0100000011  // W1197_2048 = -0.862424  +0.506187
1001001001_0100000111  // W1200_2048 = -0.857729  +0.514103
1001001011_0100001011  // W1203_2048 = -0.852961  +0.521975
1001001110_0100001111  // W1206_2048 = -0.848120  +0.529804
1001010000_0100010011  // W1209_2048 = -0.843208  +0.537587
1001010011_0100010111  // W1212_2048 = -0.838225  +0.545325
1001010101_0100011011  // W1215_2048 = -0.833170  +0.553017
1001011000_0100011111  // W1218_2048 = -0.828045  +0.560662
1001011011_0100100011  // W1221_2048 = -0.822850  +0.568259
1001011101_0100100111  // W1224_2048 = -0.817585  +0.575808
1001100000_0100101011  // W1227_2048 = -0.812251  +0.583309
1001100011_0100101110  // W1230_2048 = -0.806848  +0.590760
1001100110_0100110010  // W1233_2048 = -0.801376  +0.598161
1001101001_0100110110  // W1236_2048 = -0.795837  +0.605511
1001101011_0100111010  // W1239_2048 = -0.790230  +0.612810
1001101110_0100111101  // W1242_2048 = -0.784557  +0.620057
1001110001_0101000001  // W1245_2048 = -0.778817  +0.627252
1001110100_0101000101  // W1248_2048 = -0.773010  +0.634393
1001110111_0101001000  // W1251_2048 = -0.767139  +0.641481
1001111010_0101001100  // W1254_2048 = -0.761202  +0.648514
1001111101_0101010000  // W1257_2048 = -0.755201  +0.655493
1010000000_0101010011  // W1260_2048 = -0.749136  +0.662416
1010000100_0101010111  // W1263_2048 = -0.743008  +0.669283
1010000111_0101011010  // W1266_2048 = -0.736817  +0.676093
1010001010_0101011110  // W1269_2048 = -0.730563  +0.682846
1010001101_0101100001  // W1272_2048 = -0.724247  +0.689541
1010010000_0101100100  // W1275_2048 = -0.717870  +0.696177
1010010100_0101101000  // W1278_2048 = -0.711432  +0.702755
1010010111_0101101011  // W1281_2048 = -0.704934  +0.709273
1010011010_0101101110  // W1284_2048 = -0.698376  +0.715731
1010011110_0101110010  // W1287_2048 = -0.691759  +0.722128
1010100001_0101110101  // W1290_2048 = -0.685084  +0.728464
1010100101_0101111000  // W1293_2048 = -0.678350  +0.734739
1010101000_0101111011  // W1296_2048 = -0.671559  +0.740951
1010101100_0101111111  // W1299_2048 = -0.664711  +0.747101
1010101111_0110000010  // W1302_2048 = -0.657807  +0.753187
1010110011_0110000101  // W1305_2048 = -0.650847  +0.759209
1010110110_0110001000  // W1308_2048 = -0.643832  +0.765167
1010111010_0110001011  // W1311_2048 = -0.636762  +0.771061
1010111110_0110001110  // W1314_2048 = -0.629638  +0.776888
1011000001_0110010001  // W1317_2048 = -0.622461  +0.782651
```

1011000101_0110010100	// W1320_2048 = -0.615232	+0.788346
1011001001_0110010111	// W1323_2048 = -0.607950	+0.793975
1011001100_0110011001	// W1326_2048 = -0.600616	+0.799537
1011010000_0110011100	// W1329_2048 = -0.593232	+0.805031
1011010100_0110011111	// W1332_2048 = -0.585798	+0.810457
1011011000_0110100010	// W1335_2048 = -0.578314	+0.815814
1011011100_0110100100	// W1338_2048 = -0.570781	+0.821103
1011100000_0110100111	// W1341_2048 = -0.563199	+0.826321
1011100100_0110101010	// W1344_2048 = -0.555570	+0.831470
1011100111_0110101100	// W1347_2048 = -0.547894	+0.836548
1011101011_0110101111	// W1350_2048 = -0.540171	+0.841555
1011101111_0110110001	// W1353_2048 = -0.532403	+0.846491
1011110011_0110110100	// W1356_2048 = -0.524590	+0.851355
1011110111_0110110110	// W1359_2048 = -0.516732	+0.856147
1011111011_0110111001	// W1362_2048 = -0.508830	+0.860867
1100000000_0110111011	// W1365_2048 = -0.500885	+0.865514
1100000100_0110111101	// W1368_2048 = -0.492898	+0.870087
1100001000_0111000000	// W1371_2048 = -0.484869	+0.874587
1100001100_0111000010	// W1374_2048 = -0.476799	+0.879012
1100010000_0111000100	// W1377_2048 = -0.468689	+0.883363
1100010100_0111000110	// W1380_2048 = -0.460539	+0.887640
1100011000_0111001001	// W1383_2048 = -0.452350	+0.891841
1100011101_0111001011	// W1386_2048 = -0.444122	+0.895966
1100100001_0111001101	// W1389_2048 = -0.435857	+0.900016
1100100101_0111001111	// W1392_2048 = -0.427555	+0.903989
1100101001_0111010001	// W1395_2048 = -0.419217	+0.907886
1100101110_0111010011	// W1398_2048 = -0.410843	+0.911706
1100110010_0111010101	// W1401_2048 = -0.402435	+0.915449
1100110110_0111010111	// W1404_2048 = -0.393992	+0.919114
1100111011_0111011000	// W1407_2048 = -0.385516	+0.922701
1100111111_0111011010	// W1410_2048 = -0.377007	+0.926210
1101000011_0111011100	// W1413_2048 = -0.368467	+0.929641
1101001000_0111011110	// W1416_2048 = -0.359895	+0.932993
1101001100_0111011111	// W1419_2048 = -0.351293	+0.936266
1101010001_0111100001	// W1422_2048 = -0.342661	+0.939459
1101010101_0111100011	// W1425_2048 = -0.334000	+0.942573
1101011001_0111100100	// W1428_2048 = -0.325310	+0.945607
1101011110_0111100110	// W1431_2048 = -0.316593	+0.948561
1101100010_0111100111	// W1434_2048 = -0.307850	+0.951435
1101100111_0111101001	// W1437_2048 = -0.299080	+0.954228
1101101011_0111101010	// W1440_2048 = -0.290285	+0.956940
1101110000_0111101011	// W1443_2048 = -0.281465	+0.959572
1101110100_0111101101	// W1446_2048 = -0.272621	+0.962121
1101111001_0111101110	// W1449_2048 = -0.263755	+0.964590
1101111110_0111101111	// W1452_2048 = -0.254866	+0.966976
1110000010_0111110000	// W1455_2048 = -0.245955	+0.969281
1110000111_0111110001	// W1458_2048 = -0.237024	+0.971504
1110001011_0111110011	// W1461_2048 = -0.228072	+0.973644
1110010000_0111110100	// W1464_2048 = -0.219101	+0.975702
1110010100_0111110101	// W1467_2048 = -0.210112	+0.977677
1110011001_0111110110	// W1470_2048 = -0.201105	+0.979570
1110011110_0111110110	// W1473_2048 = -0.192080	+0.981379
1110100010_0111110111	// W1476_2048 = -0.183040	+0.983105
1110100111_0111111000	// W1479_2048 = -0.173984	+0.984749
1110101100_0111111001	// W1482_2048 = -0.164913	+0.986308
1110110000_0111111010	// W1485_2048 = -0.155828	+0.987784

```
1110110101_0111111010  // W1488_2048 = -0.146730   +0.989177
1110111010_0111111011  // W1491_2048 = -0.137620   +0.990485
1110111110_0111111100  // W1494_2048 = -0.128498   +0.991710
1111000011_0111111100  // W1497_2048 = -0.119365   +0.992850
1111001000_0111111101  // W1500_2048 = -0.110222   +0.993907
1111001100_0111111101  // W1503_2048 = -0.101070   +0.994879
1111010001_0111111110  // W1506_2048 = -0.091909   +0.995767
1111010110_0111111110  // W1509_2048 = -0.082740   +0.996571
1111011010_0111111111  // W1512_2048 = -0.073565   +0.997290
1111011111_0111111111  // W1515_2048 = -0.064383   +0.997925
1111100100_0111111111  // W1518_2048 = -0.055195   +0.998476
1111101000_0111111111  // W1521_2048 = -0.046003   +0.998941
1111101101_0111111111  // W1524_2048 = -0.036807   +0.999322
1111110010_0111111111  // W1527_2048 = -0.027608   +0.999619
1111110111_0111111111  // W1530_2048 = -0.018407   +0.999831
1111111011_0111111111  // W1533_2048 = -0.009204   +0.999958
```

Listing 17

```
// 512 point FFT twiddle factor coefficients (Radix 4+2).
// Coefficients stored as non-fractional 10 bit integers (scale 1 ).
// Real Coefficient (cosine value) is coefficient high-byte.
// Imaginary Coefficient (sine value) is coefficient low-byte.

0111111111_0000000000  // W0000_0512 = +1.000000   -0.000000
0111111111_1111111010  // W0001_0512 = +0.999925   -0.012272
0111111111_1111110011  // W0002_0512 = +0.999699   -0.024541
0111111111_1111101101  // W0003_0512 = +0.999322   -0.036807
0111111111_1111100111  // W0004_0512 = +0.998795   -0.049068
0111111111_1111100001  // W0005_0512 = +0.998118   -0.061321
0111111111_1111011010  // W0006_0512 = +0.997290   -0.073565
0111111110_1111010100  // W0007_0512 = +0.996313   -0.085797
0111111110_1111001110  // W0008_0512 = +0.995185   -0.098017
0111111101_1111001000  // W0009_0512 = +0.993907   -0.110222
0111111100_1111000001  // W0010_0512 = +0.992480   -0.122411
0111111011_1110111011  // W0011_0512 = +0.990903   -0.134581
0111111010_1110110101  // W0012_0512 = +0.989177   -0.146730
0111111001_1110101111  // W0013_0512 = +0.987301   -0.158858
0111111000_1110101000  // W0014_0512 = +0.985278   -0.170962
0111110111_1110100010  // W0015_0512 = +0.983105   -0.183040
0111110110_1110011100  // W0016_0512 = +0.980785   -0.195090
0111110101_1110010110  // W0017_0512 = +0.978317   -0.207111
0111110100_1110010000  // W0018_0512 = +0.975702   -0.219101
0111110010_1110001010  // W0019_0512 = +0.972940   -0.231058
0111110001_1110000100  // W0020_0512 = +0.970031   -0.242980
0111101111_1101111110  // W0021_0512 = +0.966976   -0.254866
0111101101_1101110111  // W0022_0512 = +0.963776   -0.266713
0111101100_1101110001  // W0023_0512 = +0.960431   -0.278520
0111101010_1101101011  // W0024_0512 = +0.956940   -0.290285
0111101000_1101100101  // W0025_0512 = +0.953306   -0.302006
0111100110_1101011111  // W0026_0512 = +0.949528   -0.313682
0111100100_1101011001  // W0027_0512 = +0.945607   -0.325310
0111100010_1101010100  // W0028_0512 = +0.941544   -0.336890
0111100000_1101001110  // W0029_0512 = +0.937339   -0.348419
0111011110_1101001000  // W0030_0512 = +0.932993   -0.359895
0111011011_1101000010  // W0031_0512 = +0.928506   -0.371317
```

```
0111011001_1100111100  // W0032_0512 = +0.923880   -0.382683
0111010111_1100110110  // W0033_0512 = +0.919114   -0.393992
0111010100_1100110001  // W0034_0512 = +0.914210   -0.405241
0111010001_1100101011  // W0035_0512 = +0.909168   -0.416430
0111001111_1100100101  // W0036_0512 = +0.903989   -0.427555
0111001100_1100011111  // W0037_0512 = +0.898674   -0.438616
0111001001_1100011010  // W0038_0512 = +0.893224   -0.449611
0111000110_1100010100  // W0039_0512 = +0.887640   -0.460539
0111000100_1100001111  // W0040_0512 = +0.881921   -0.471397
0111000001_1100001001  // W0041_0512 = +0.876070   -0.482184
0110111101_1100000100  // W0042_0512 = +0.870087   -0.492898
0110111010_1011111110  // W0043_0512 = +0.863973   -0.503538
0110110111_1011111001  // W0044_0512 = +0.857729   -0.514103
0110110100_1011110011  // W0045_0512 = +0.851355   -0.524590
0110110001_1011101110  // W0046_0512 = +0.844854   -0.534998
0110101101_1011101001  // W0047_0512 = +0.838225   -0.545325
0110101010_1011100100  // W0048_0512 = +0.831470   -0.555570
0110100110_1011011110  // W0049_0512 = +0.824589   -0.565732
0110100011_1011011001  // W0050_0512 = +0.817585   -0.575808
0110011111_1011010100  // W0051_0512 = +0.810457   -0.585798
0110011011_1011001111  // W0052_0512 = +0.803208   -0.595699
0110010111_1011001010  // W0053_0512 = +0.795837   -0.605511
0110010100_1011000101  // W0054_0512 = +0.788346   -0.615232
0110010000_1011000000  // W0055_0512 = +0.780737   -0.624859
0110001100_1010111011  // W0056_0512 = +0.773010   -0.634393
0110001000_1010110110  // W0057_0512 = +0.765167   -0.643832
0110000100_1010110010  // W0058_0512 = +0.757209   -0.653173
0110000000_1010101101  // W0059_0512 = +0.749136   -0.662416
0101111011_1010101000  // W0060_0512 = +0.740951   -0.671559
0101110111_1010100100  // W0061_0512 = +0.732654   -0.680601
0101110011_1010011111  // W0062_0512 = +0.724247   -0.689541
0101101110_1010011010  // W0063_0512 = +0.715731   -0.698376
0101101010_1010010110  // W0064_0512 = +0.707107   -0.707107
0101100110_1010010010  // W0065_0512 = +0.698376   -0.715731
0101100001_1010001101  // W0066_0512 = +0.689541   -0.724247
0101011100_1010001001  // W0067_0512 = +0.680601   -0.732654
0101011000_1010000101  // W0068_0512 = +0.671559   -0.740951
0101010011_1010000000  // W0069_0512 = +0.662416   -0.749136
0101001110_1001111100  // W0070_0512 = +0.653173   -0.757209
0101001010_1001111000  // W0071_0512 = +0.643832   -0.765167
0101000101_1001110100  // W0072_0512 = +0.634393   -0.773010
0101000000_1001110000  // W0073_0512 = +0.624859   -0.780737
0100111011_1001101100  // W0074_0512 = +0.615232   -0.788346
0100110110_1001101001  // W0075_0512 = +0.605511   -0.795837
0100110001_1001100101  // W0076_0512 = +0.595699   -0.803208
0100101100_1001100001  // W0077_0512 = +0.585798   -0.810457
0100100111_1001011101  // W0078_0512 = +0.575808   -0.817585
0100100010_1001011010  // W0079_0512 = +0.565732   -0.824589
0100011100_1001010110  // W0080_0512 = +0.555570   -0.831470
0100010111_1001010011  // W0081_0512 = +0.545325   -0.838225
0100010010_1001001111  // W0082_0512 = +0.534998   -0.844854
0100001101_1001001100  // W0083_0512 = +0.524590   -0.851355
0100000111_1001001001  // W0084_0512 = +0.514103   -0.857729
0100000010_1001000110  // W0085_0512 = +0.503538   -0.863973
0011111100_1001000011  // W0086_0512 = +0.492898   -0.870087
0011110111_1000111111  // W0087_0512 = +0.482184   -0.876070
```

```
0011110001_1000111100 // W0088_0512 = +0.471397 -0.881921
0011101100_1000111010 // W0089_0512 = +0.460539 -0.887640
0011100110_1000110111 // W0090_0512 = +0.449611 -0.893224
0011100001_1000110100 // W0091_0512 = +0.438616 -0.898674
0011011011_1000110001 // W0092_0512 = +0.427555 -0.903989
0011010101_1000101111 // W0093_0512 = +0.416430 -0.909168
0011001111_1000101100 // W0094_0512 = +0.405241 -0.914210
0011001010_1000101001 // W0095_0512 = +0.393992 -0.919114
0011000100_1000100111 // W0096_0512 = +0.382683 -0.923880
0010111110_1000100101 // W0097_0512 = +0.371317 -0.928506
0010111000_1000100010 // W0098_0512 = +0.359895 -0.932993
0010110010_1000100000 // W0099_0512 = +0.348419 -0.937339
0010101100_1000011110 // W0100_0512 = +0.336890 -0.941544
0010100111_1000011100 // W0101_0512 = +0.325310 -0.945607
0010100001_1000011010 // W0102_0512 = +0.313682 -0.949528
0010011011_1000011000 // W0103_0512 = +0.302006 -0.953306
0010010101_1000010110 // W0104_0512 = +0.290285 -0.956940
0010001111_1000010100 // W0105_0512 = +0.278520 -0.960431
0010001001_1000010011 // W0106_0512 = +0.266713 -0.963776
0010000010_1000010001 // W0107_0512 = +0.254866 -0.966976
0001111100_1000001111 // W0108_0512 = +0.242980 -0.970031
0001110110_1000001110 // W0109_0512 = +0.231058 -0.972940
0001110000_1000001100 // W0110_0512 = +0.219101 -0.975702
0001101010_1000001011 // W0111_0512 = +0.207111 -0.978317
0001100100_1000001010 // W0112_0512 = +0.195090 -0.980785
0001011110_1000001001 // W0113_0512 = +0.183040 -0.983105
0001011000_1000001000 // W0114_0512 = +0.170962 -0.985278
0001010001_1000000111 // W0115_0512 = +0.158858 -0.987301
0001001011_1000000110 // W0116_0512 = +0.146730 -0.989177
0001000101_1000000101 // W0117_0512 = +0.134581 -0.990903
0000111111_1000000100 // W0118_0512 = +0.122411 -0.992480
0000111000_1000000011 // W0119_0512 = +0.110222 -0.993907
0000110010_1000000010 // W0120_0512 = +0.098017 -0.995185
0000101100_1000000010 // W0121_0512 = +0.085797 -0.996313
0000100110_1000000001 // W0122_0512 = +0.073565 -0.997290
0000011111_1000000001 // W0123_0512 = +0.061321 -0.998118
0000011001_1000000001 // W0124_0512 = +0.049068 -0.998795
0000010011_1000000000 // W0125_0512 = +0.036807 -0.999322
0000001101_1000000000 // W0126_0512 = +0.024541 -0.999699
0000000110_1000000000 // W0127_0512 = +0.012272 -0.999925
0000000000_1000000000 // W0128_0512 = +0.000000 -1.000000
1111111010_1000000000 // W0129_0512 = -0.012272 -0.999925
1111110011_1000000000 // W0130_0512 = -0.024541 -0.999699
1111100111_1000000001 // W0132_0512 = -0.049068 -0.998795
1111011010_1000000001 // W0134_0512 = -0.073565 -0.997290
1111010100_1000000010 // W0135_0512 = -0.085797 -0.996313
1111001110_1000000010 // W0136_0512 = -0.098017 -0.995185
1111000001_1000000100 // W0138_0512 = -0.122411 -0.992480
1110110101_1000000110 // W0140_0512 = -0.146730 -0.989177
1110101111_1000000111 // W0141_0512 = -0.158858 -0.987301
1110101000_1000001000 // W0142_0512 = -0.170962 -0.985278
1110011100_1000001010 // W0144_0512 = -0.195090 -0.980785
1110010000_1000001100 // W0146_0512 = -0.219101 -0.975702
1110001010_1000001110 // W0147_0512 = -0.231058 -0.972940
1110000100_1000001111 // W0148_0512 = -0.242980 -0.970031
1101110111_1000010011 // W0150_0512 = -0.266713 -0.963776
```

```
1101101011_1000010110  // W0152_0512 = -0.290285   -0.956940
1101100101_1000011000  // W0153_0512 = -0.302006   -0.953306
1101011111_1000011010  // W0154_0512 = -0.313682   -0.949528
1101010100_1000011110  // W0156_0512 = -0.336890   -0.941544
1101001000_1000100010  // W0158_0512 = -0.359895   -0.932993
1101000010_1000100101  // W0159_0512 = -0.371317   -0.928506
1100111100_1000100111  // W0160_0512 = -0.382683   -0.923880
1100110001_1000101100  // W0162_0512 = -0.405241   -0.914210
1100100101_1000110001  // W0164_0512 = -0.427555   -0.903989
1100011111_1000110100  // W0165_0512 = -0.438616   -0.898674
1100011010_1000110111  // W0166_0512 = -0.449611   -0.893224
1100001111_1000111100  // W0168_0512 = -0.471397   -0.881921
1100000100_1001000011  // W0170_0512 = -0.492898   -0.870087
1011111110_1001000110  // W0171_0512 = -0.503538   -0.863973
1011111001_1001001001  // W0172_0512 = -0.514103   -0.857729
1011101110_1001001111  // W0174_0512 = -0.534998   -0.844854
1011100100_1001010110  // W0176_0512 = -0.555570   -0.831470
1011011110_1001011010  // W0177_0512 = -0.565732   -0.824589
1011011001_1001011101  // W0178_0512 = -0.575808   -0.817585
1011001111_1001100101  // W0180_0512 = -0.595699   -0.803208
1011000101_1001101100  // W0182_0512 = -0.615232   -0.788346
1011000000_1001110000  // W0183_0512 = -0.624859   -0.780737
1010111011_1001110100  // W0184_0512 = -0.634393   -0.773010
1010110010_1001111100  // W0186_0512 = -0.653173   -0.757209
1010101000_1010000101  // W0188_0512 = -0.671559   -0.740951
1010100100_1010001001  // W0189_0512 = -0.680601   -0.732654
1010011111_1010001101  // W0190_0512 = -0.689541   -0.724247
1010010110_1010010110  // W0192_0512 = -0.707107   -0.707107
1010001101_1010011111  // W0194_0512 = -0.724247   -0.689541
1010001001_1010100100  // W0195_0512 = -0.732654   -0.680601
1010000101_1010101000  // W0196_0512 = -0.740951   -0.671559
1001111100_1010110010  // W0198_0512 = -0.757209   -0.653173
1001110100_1010111011  // W0200_0512 = -0.773010   -0.634393
1001110000_1011000000  // W0201_0512 = -0.780737   -0.624859
1001101100_1011000101  // W0202_0512 = -0.788346   -0.615232
1001100101_1011001111  // W0204_0512 = -0.803208   -0.595699
1001011101_1011011001  // W0206_0512 = -0.817585   -0.575808
1001011010_1011011110  // W0207_0512 = -0.824589   -0.565732
1001010110_1011100100  // W0208_0512 = -0.831470   -0.555570
1001001111_1011101110  // W0210_0512 = -0.844854   -0.534998
1001001001_1011111001  // W0212_0512 = -0.857729   -0.514103
1001000110_1011111110  // W0213_0512 = -0.863973   -0.503538
1001000011_1100000100  // W0214_0512 = -0.870087   -0.492898
1000111100_1100001111  // W0216_0512 = -0.881921   -0.471397
1000110111_1100011010  // W0218_0512 = -0.893224   -0.449611
1000110100_1100011111  // W0219_0512 = -0.898674   -0.438616
1000110001_1100100101  // W0220_0512 = -0.903989   -0.427555
1000101100_1100110001  // W0222_0512 = -0.914210   -0.405241
1000100111_1100111100  // W0224_0512 = -0.923880   -0.382683
1000100101_1101000010  // W0225_0512 = -0.928506   -0.371317
1000100010_1101001000  // W0226_0512 = -0.932993   -0.359895
1000011110_1101010100  // W0228_0512 = -0.941544   -0.336890
1000011010_1101011111  // W0230_0512 = -0.949528   -0.313682
1000011000_1101100101  // W0231_0512 = -0.953306   -0.302006
1000010110_1101101011  // W0232_0512 = -0.956940   -0.290285
1000010011_1101110111  // W0234_0512 = -0.963776   -0.266713
```

```
1000001111_1110000100  // W0236_0512 = -0.970031   -0.242980
1000001110_1110001010  // W0237_0512 = -0.972940   -0.231058
1000001100_1110010000  // W0238_0512 = -0.975702   -0.219101
1000001010_1110011100  // W0240_0512 = -0.980785   -0.195090
1000001000_1110101000  // W0242_0512 = -0.985278   -0.170962
1000000111_1110101111  // W0243_0512 = -0.987301   -0.158858
1000000110_1110110101  // W0244_0512 = -0.989177   -0.146730
1000000100_1111000001  // W0246_0512 = -0.992480   -0.122411
1000000010_1111001110  // W0248_0512 = -0.995185   -0.098017
1000000010_1111010100  // W0249_0512 = -0.996313   -0.085797
1000000001_1111011010  // W0250_0512 = -0.997290   -0.073565
1000000001_1111100111  // W0252_0512 = -0.998795   -0.049068
1000000000_1111110011  // W0254_0512 = -0.999699   -0.024541
1000000000_1111111010  // W0255_0512 = -0.999925   -0.012272
1000000000_0000001101  // W0258_0512 = -0.999699   +0.024541
1000000001_0000011111  // W0261_0512 = -0.998118   +0.061321
1000000010_0000110010  // W0264_0512 = -0.995185   +0.098017
1000000101_0001000101  // W0267_0512 = -0.990903   +0.134581
1000001000_0001011000  // W0270_0512 = -0.985278   +0.170962
1000001011_0001101010  // W0273_0512 = -0.978317   +0.207111
1000001111_0001111100  // W0276_0512 = -0.970031   +0.242980
1000010100_0010001111  // W0279_0512 = -0.960431   +0.278520
1000011010_0010100001  // W0282_0512 = -0.949528   +0.313682
1000100000_0010110010  // W0285_0512 = -0.937339   +0.348419
1000100111_0011000100  // W0288_0512 = -0.923880   +0.382683
1000101111_0011010101  // W0291_0512 = -0.909168   +0.416430
1000110111_0011100110  // W0294_0512 = -0.893224   +0.449611
1000111111_0011110111  // W0297_0512 = -0.876070   +0.482184
1001001001_0100000111  // W0300_0512 = -0.857729   +0.514103
1001010011_0100010111  // W0303_0512 = -0.838225   +0.545325
1001011101_0100100111  // W0306_0512 = -0.817585   +0.575808
1001101001_0100110110  // W0309_0512 = -0.795837   +0.605511
1001110100_0101000101  // W0312_0512 = -0.773010   +0.634393
1010000000_0101010011  // W0315_0512 = -0.749136   +0.662416
1010001101_0101100001  // W0318_0512 = -0.724247   +0.689541
1010011010_0101101110  // W0321_0512 = -0.698376   +0.715731
1010101000_0101111011  // W0324_0512 = -0.671559   +0.740951
1010110110_0110001000  // W0327_0512 = -0.643832   +0.765167
1011000101_0110010100  // W0330_0512 = -0.615232   +0.788346
1011010100_0110011111  // W0333_0512 = -0.585798   +0.810457
1011100100_0110101010  // W0336_0512 = -0.555570   +0.831470
1011110011_0110110100  // W0339_0512 = -0.524590   +0.851355
1100000100_0110111101  // W0342_0512 = -0.492898   +0.870087
1100010100_0111000110  // W0345_0512 = -0.460539   +0.887640
1100100101_0111001111  // W0348_0512 = -0.427555   +0.903989
1100110110_0111010111  // W0351_0512 = -0.393992   +0.919114
1101001000_0111011110  // W0354_0512 = -0.359895   +0.932993
1101011001_0111100100  // W0357_0512 = -0.325310   +0.945607
1101101011_0111101010  // W0360_0512 = -0.290285   +0.956940
1101111110_0111101111  // W0363_0512 = -0.254866   +0.966976
1110010000_0111110100  // W0366_0512 = -0.219101   +0.975702
1110100010_0111110111  // W0369_0512 = -0.183040   +0.983105
1110110101_0111111010  // W0372_0512 = -0.146730   +0.989177
1111001000_0111111101  // W0375_0512 = -0.110222   +0.993907
1111011010_0111111111  // W0378_0512 = -0.073565   +0.997290
1111101101_0111111111  // W0381_0512 = -0.036807   +0.999322
```

Listing 18

```
/*FOLDBEGINS 0 0 "Copyright"*/
/**********************************************************
Copyright (c) Pioneer Digital Design Centre Limited

NAME:  pilloc_rtl.v

PURPOSE: Pilot location

CREATED:    June 1997  BY: T. Foxcroft

MODIFIED:

USED IN PROJECTS:  cofdm only.

**********************************************************/
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "Defines"*/
`define FFTSIZE    2048
`define DATABINS   1705
`define SCATNUM    45
`define SCALEFACTOR64Q 3792   //3x8192/sqrt(42)
`define SCALEFACTOR16Q 3886   //3x8192/sqrt(10)*2
`define SCALEFACTORQPS 2172   //3x8192/sqrt(2)*8
`define AVERAGESF    12'hc49 //0.04x4096x32768/1705 = 3145
/*FOLDENDS*/
module chanest (clk, resync, in_valid, in_data, constellation,
                u_symbol, us_pilots, uc_pilots, ct_pilots, out_tps, tps_valid,
                uncorrected_iq,
                out_valid, outi, outq, c_symbol, incfreq, wrstrb, ramindata,
                ramoutdata, ramaddr);
/*FOLDBEGINS 0 0 "i/o"*/
input clk, resync, in_valid;
input [23:0] in_data;
input [1:0] constellation;
output u_symbol;
output us_pilots, uc_pilots, ct_pilots;
output out_tps, tps_valid;
output [23:0] uncorrected_iq;
output out_valid;
output [7:0] outi;
output [7:0] outq;
output c_symbol;
output incfreq;
output wrstrb;
output [23:0] ramindata;
input [23:0] ramoutdata;
output [10:0] ramaddr;

/*FOLDENDS*/
/*FOLDBEGINS 0 0 "TPS location "*/
reg [10:0] tpsloc;
reg [4:0] tpscount;
```

```
always @(tpscount)
begin
   case(tpscount)
   5'b00000: tpsloc = 34;
   5'b00001: tpsloc = 50;
   5'b00010: tpsloc = 209;
   5'b00011: tpsloc = 346;
   5'b00100: tpsloc = 413;
   5'b00101: tpsloc = 569;
   5'b00110: tpsloc = 595;
   5'b00111: tpsloc = 688;
   5'b01000: tpsloc = 790;
   5'b01001: tpsloc = 901;
   5'b01010: tpsloc = 1073;
   5'b01011: tpsloc = 1219;
   5'b01100: tpsloc = 1262;
   5'b01101: tpsloc = 1286;
   5'b01110: tpsloc = 1469;
   5'b01111: tpsloc = 1594;
   default: tpsloc = 1687;
   endcase
end
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "continuous pilot location"*/
reg [10:0] contloc;
reg [5:0] contloccount;
always @(contloccount)
begin
   case(contloccount)
   6'b000000: contloc = 0;
   6'b000001: contloc = 48;
   6'b000010: contloc = 54;
   6'b000011: contloc = 87;
   6'b000100: contloc = 141;
   6'b000101: contloc = 156;
   6'b000110: contloc = 192;
   6'b000111: contloc = 201;
   6'b001000: contloc = 255;
   6'b001001: contloc = 279;
   6'b001010: contloc = 282;
   6'b001011: contloc = 333;
   6'b001100: contloc = 432;
   6'b001101: contloc = 450;
   6'b001110: contloc = 483;
   6'b001111: contloc = 525;
   6'b010000: contloc = 531;
   6'b010001: contloc = 618;
   6'b010010: contloc = 636;
   6'b010011: contloc = 714;
   6'b010100: contloc = 759;
   6'b010101: contloc = 765;
   6'b010110: contloc = 780;
   6'b010111: contloc = 804;
   6'b011000: contloc = 873;
   6'b011001: contloc = 888;
   6'b011010: contloc = 918;
```

```
6'b011011: contloc = 939;
6'b011100: contloc = 942;
6'b011101: contloc = 969;
6'b011110: contloc = 984;
6'b011111: contloc = 1050;
6'b100000: contloc = 1101;
6'b100001: contloc = 1107;
6'b100010: contloc = 1110;
6'b100011: contloc = 1137;
6'b100100: contloc = 1140;
6'b100101: contloc = 1146;
6'b100110: contloc = 1206;
6'b100111: contloc = 1269;
6'b101000: contloc = 1323;
6'b101001: contloc = 1377;
6'b101010: contloc = 1491;
6'b101011: contloc = 1683;
default: contloc = 1704;
endcase
end
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "continuous pilot location"*/
/*reg [10:0] contloc [44:0];
reg [5:0] contloccount;
initial
begin
  contloc[0] = 0; contloc[1] = 48; contloc[2] = 54; contloc[3] = 87; contloc[4] = 141;
  contloc[5] = 156; contloc[6] = 192; contloc[7] = 201; contloc[8] = 255; contloc[9] =
  279;
  contloc[10] = 282; contloc[11] = 333; contloc[12] = 432; contloc[13] = 450;
  contloc[14] = 483;
  contloc[15] = 525; contloc[16] = 531; contloc[17] = 618; contloc[18] = 636;
  contloc[19] = 714;
  contloc[20] = 759; contloc[21] = 765; contloc[22] = 780; contloc[23] = 804;
  contloc[24] = 873;
  contloc[25] = 888; contloc[26] = 918; contloc[27] = 939; contloc[28] = 942;
  contloc[29] = 969;
  contloc[30] = 984; contloc[31] = 1050; contloc[32] = 1101; contloc[33] = 1107;
  contloc[34] = 1110;
  contloc[35] = 1137; contloc[36] = 1140; contloc[37] = 1146; contloc[38] = 1206;
  contloc[39] = 1269;
  contloc[40] = 1323; contloc[41] = 1377; contloc[42] = 1491; contloc[43] = 1683;
  contloc[44] = 1704;
end */
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "Control vars"*/
reg [1:0] constell;
reg resynch;
reg valid,valid0,valid1,valid2,valid3,valid4,valid5,valid6,valid7,valid8;
reg [1:0] whichsymbol;
reg [1:0] pwhichsymbol;
reg incwhichsymbol;
reg [23:0] fftdata;
reg [10:0] fftcount;
reg [10:0] tapcount;
reg [3:0] count12;
```

```
reg [3:0] dcount12;
reg ramdatavalid;
reg tapinit;
reg tapinit1,tapinit2;
reg [7:0] nscat;
reg pilot;
reg tapload; //controls when the taps are loaded
reg tapload2;
reg shiftinnewtap;
reg filtgo;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "Channel Est vars"*/
reg [11:0] tapi [5:0];
reg [11:0] tapq [5:0];
reg [27:0] sumi;
reg [27:0] sumq;
reg [11:0] chani;
reg [11:0] chanq;
wire [27:0] chani_;
wire [27:0] chanq_;
reg [11:0] idata;
reg [11:0] qdata;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "RAM vars"*/
reg [10:0] ramaddr;
reg [10:0] pilotaddr;
wire [10:0] ramaddr_;
wire [10:0] ramaddrrev_;
reg [23:0] ramindata;
wire [23:0] ramoutdata;
reg [23:0] ramout;
reg [23:0] ramot;
reg wrstrb;
reg rwtoggle;
reg framedata, framedata0;
reg frav, firstfrav;
reg [23:0] avchannel;
reg [11:0] avchan;
reg avlow;
wire [23:0] avchan_;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "Channel calc vars"*/
reg chan_val;
reg chan_val0,chan_val1,chan_val2,chan_val3,chan_val4,out_valid;
reg [23:0] sum;
reg [11:0] sumsq;
reg [11:0] sumsqtemp;
reg [11:0] topreal;
reg [11:0] topimag;
reg [7:0] outi;
reg [7:0] outitemp;
reg [5:0] outitem;
reg [7:0] outq;
reg [10:0] prbs;
//integer intsumi, intsumq,intsumsq,intouti,intoutq;
/*FOLDENDS*/
```

```
/*FOLDBEGINS 0 0 "uncorrected pilot vars"*/
reg u_symbol;
reg us_pilots;
reg uc_pilots;
reg [23:0] uncorrected_iq;
reg [2:0] tps_pilots;
reg [5:0] tpsmajcount;
wire [5:0] tpsmajcount_;
reg ct_pilots;
reg out_tps, tps_valid;
reg [1:0] pilotdata;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "pilot locate vars"*/
wire [1:0] which_symbol;
wire [10:0] cpoffset;
wire [10:0] pilotramaddr_;
wire [23:0] pilotramin_;
wire pilotwrstrb_;
wire found_pilots;
reg  pilotlocated;

/*FOLDENDS*/
/*FOLDBEGINS 0 0 "sync function arrays"*/
reg [11:0] sync0;
reg [11:0] sync1;
reg [11:0] sync2;
reg [3:0] syncoffset;
always @(dcount12 or valid1 or valid2)
begin
   if(valid1 || valid2)
   syncoffset = 4'hc-dcount12;
   else
   syncoffset = dcount12;
/*FOLDBEGINS 0 2 ""*/
case(syncoffset)
4'h1:
begin
      sync0 = 4046; sync1 = 272; sync2 = 95;
      end
      4'h2:
      begin
      sync0 = 3899; sync1 = 476; sync2 = 168;
      end
      4'h3:
      begin
      sync0 = 3661; sync1 = 614; sync2 = 217;
      end
      4'h4:
      begin
      sync0 = 3344; sync1 = 687; sync2 = 243;
      end
      4'h5:
      begin
      sync0 = 2963; sync1 = 701; sync2 = 248;
      end
      4'h6:
```

```
        begin
        sync0 = 2534; sync1 = 665; sync2 = 234;
        end
        4'h7:
        begin
        sync0 = 2076; sync1 = 590; sync2 = 205;
        end
        4'h8:
        begin
        sync0 = 1609; sync1 = 486; sync2 = 167;
        end
        4'h9:
        begin
        sync0 = 1152; sync1 = 364; sync2 = 123;

        end
        4'ha:
        begin
        sync0 = 722;  sync1 = 237; sync2 = 78;
        end
        default
        begin
        sync0 = 334;  sync1 = 113; sync2 = 36;
        end
        endcase
        /*FOLDENDS*/
    end
    /*FOLDENDS*/
    always @(posedge clk)
    begin
    /*FOLDBEGINS 0 2 "Control "*/
       constell <= constellation;
       resynch <= resync;
       if(resynch)
       begin
       /*FOLDBEGINS 0 2 ""*/
          valid     <= 1'b0;
          valid0    <= 1'b0;
          valid1    <= 1'b0;
          valid2    <= 1'b0;
          valid3    <= 1'b0;
          valid4    <= 1'b0;
          valid5    <= 1'b0;
          valid6    <= 1'b0;
          valid7    <= 1'b0;
          valid8    <= 1'b0;
          fftcount   <= 11'b0;
          ramdatavalid <= 1'b0;
          chan_val   <= 1'b0;
          tapinit    <= 1'b0;
          tapinit1   <= 1'b0;
          tapinit2   <= 1'b0;
          rwtoggle   <= 1'b0;
          /*FOLDENDS*/
       end
       else
```

```
begin
/*FOLDBEGINS 0 2 ""*/
   valid <= in_valid;
   valid0 <= valid&&pilotlocated;
   valid1 <= valid0;
   valid2 <= valid1;
   valid3 <= valid2;
   valid4 <= valid3;
   valid5 <= valid4;
   valid6 <= valid5;
   valid7 <= valid6;
   valid8 <= valid7;
   if(valid2)

      fftcount <= fftcount + 1'b1;
      chan_val <= valid4&&filtgo&&framedata;
      incwhichsymbol <= valid1&&(fftcount == (`FFTSIZE-1));
      if(incwhichsymbol)
      begin
      rwtoggle  <= !rwtoggle;
      tapinit <= 1'b1;
      ramdatavalid <= 1'b1;
   end
   else if(valid6)
      tapinit <= 1'b0;


   tapinit1 <= tapinit;
   tapinit2 <= tapinit1;
   /*FOLDENDS*/
end
fftdata <= in_data;
/*FOLDBEGINS 0 0 "frame averager"*/
if(resynch)
begin
   frav    <= 1'b0;
   firstfrav <= 1'b0;
end
else
begin
   if(chan_val&&framedata)
   frav <= 1'b1;
   else if(!framedata&&framedata0)
   frav <= 1'b0;
   if(chan_val&&framedata&&!frav)
   firstfrav <= 1'b1;
   else if(chan_val)
   firstfrav <= 1'b0;
/*FOLDBEGINS 0 2 "calculate 0.2 x mean channel amplitude"*/
if(chan_val0)
begin
      if(firstfrav)
      begin
         avchannel <= avmult(sumsqtemp);
         avchan  <= avchan_[11:0];
      end
```

```
        else
            avchannel <= avmult(sumsqtemp) + avchannel;
            end
            /*FOLDENDS*/
            if(chan_val1)
        avlow <= (sumsqtemp<avchan)? 1:0;

end
/*FOLDENDS*/
if(resynch)
begin
    framedata   <= 1'b0;

    framedata0  <= 1'b0;
    tapload    <= 1'b0;
end
else
begin
    framedata0 <= framedata;
    if(incwhichsymbol&&(cpoffset==0))
        framedata <= 1;
        else if(ramdatavalid&&valid2&&(fftcount == (cpoffset - 1)))
        framedata <= 1;
        else if(valid2&&(fftcount == (cpoffset + `DATABINS)))
        framedata <= 0;
        tapload <= framedata;
end
filtgo <= ramdatavalid&&( valid2? tapload : filtgo);
tapload2 <= valid&&tapload&&(count12==11)&&(fftcount!=0);
pilot <= (count12==0);
dcount12 <= count12;
shiftinnewtap <= !((nscat == 139)||(nscat == 140)||(nscat == 141));

if(incwhichsymbol)
begin
    if(!ramdatavalid)
    begin
        whichsymbol  <= pwhichsymbol;
        tapcount   <= pwhichsymbol*2'b11 + cpoffset;
    end
    else
    begin
        whichsymbol <= whichsymbol + 1'b1;
        tapcount      <=   {whichsymbol[1]^whichsymbol[0],!whichsymbol[0]}*2'b11  +
        cpoffset;
    end
    end
    else
    if(framedata)
    begin
    if(fftcount==cpoffset)
    begin
 /*FOLDBEGINS 0 4 "set up the counters"*/
 //count12 <= ((4-whichsymbol)&4'b0011)*3;
 count12 <= {whichsymbol[1]^whichsymbol[0],whichsymbol[0]}*2'b11;
 if(valid0)
```

```
                nscat  <= 8'b0;
                /*FOLDENDS*/
            end
            else
            begin
        /*FOLDBEGINS 0 4 ""*/
        if(valid)
        begin
                    count12 <= (count12==11)? 4'b0 : count12 + 1'b1;
                    tapcount <= tapcount + 1'b1;
                    if(count12==11)

                        nscat  <= nscat + 1'b1;
                        end
            /*FOLDENDS*/
            end
        end
        else
        begin
            if(tapinit2&&valid5)
            nscat  <= 8'b0;
            if(tapinit)
            begin
                if(valid3||valid4||valid5&&(whichsymbol==2'b0))
                tapcount <= tapcount + 4'hc;
                else
                if(valid6)
                        tapcount <= tapcount +
            {whichsymbol[1]^whichsymbol[0],whichsymbol[0]}*2'b11 + 1'b1;
                        end
        end
        /*FOLDENDS*/
    /*FOLDBEGINS 0 2 "Channel Estimation"*/
    if(tapinit2)
    begin
        /*FOLDBEGINS 0 4 "Read in first 3 or 4 taps"*/
        if(valid5)
                prbs   <= alpha12(alpha(whichsymbol));
                else
                if(valid6||valid7||(valid8&&(whichsymbol==2'b0)))
                prbs   <= alpha12(prbs);
                if(valid5)
                begin
                tapi[0] <= pseudo(ramout[23:12],1'b1);
                tapi[1] <= pseudo(ramout[23:12],1'b1);
                tapi[2] <= pseudo(ramout[23:12],1'b1);
                tapi[3] <= pseudo(ramout[23:12],1'b1);
                tapq[0] <= pseudo(ramout[11:0], 1'b1);
                tapq[1] <= pseudo(ramout[11:0], 1'b1);
                tapq[2] <= pseudo(ramout[11:0], 1'b1);
                tapq[3] <= pseudo(ramout[11:0], 1'b1);
            end
            else if( !((whichsymbol!=2'b0)&&valid8))
            begin
            tapi[5] <= tapi[4];
            tapi[4] <= tapi[3];
```

```
tapi[3] <= tapi[2];
tapi[2] <= tapi[1];
tapi[1] <= tapi[0];
tapi[0] <= pseudo(ramout[23:12],prbs[0]);
tapq[5] <= tapq[4];
tapq[4] <= tapq[3];
tapq[3] <= tapq[2];
tapq[2] <= tapq[1];
tapq[1] <= tapq[0];
tapq[0] <= pseudo(ramout[11:0],prbs[0]);

end
/*FOLDENDS*/
end
else if(framedata)
begin
/*FOLDBEGINS 0 4 "update taps in normal op."*/
if(tapload2)
begin
    prbs   <= alpha12(prbs);
    tapi[5] <= tapi[4];
    tapi[4] <= tapi[3];
    tapi[3] <= tapi[2];
    tapi[2] <= tapi[1];
    tapi[1] <= tapi[0];
    if(shiftinnewtap)
      tapi[0] <= pseudo(ramout[23:12],prbs[0]);
      tapq[5] <= tapq[4];
      tapq[4] <= tapq[3];
      tapq[3] <= tapq[2];
      tapq[2] <= tapq[1];
      tapq[1] <= tapq[0];
      if(shiftinnewtap)
      tapq[0] <= pseudo(ramout[11:0],prbs[0]);
      end
      /*FOLDENDS*/
/*FOLDBEGINS 0 4 "Channel interpolate"*/
if(pilot)
begin
    if(valid4)
    begin
      chani <= tapi[3];
      chanq <= tapq[3];
    end
    if(valid3)
    begin
      idata <= ramot[23:12];
      qdata <= ramot[11:0];
    end
    end
    else
    begin
    if(valid1)
    begin
      sumi <=    mult(tapi[0],sync2) - mult(tapi[1],sync1);
      sumq <=    mult(tapq[0],sync2);
```

```
			end
			else if(valid2)
			begin
				sumi <=  sumi + mult(tapi[2],sync0);
				sumq <=  sumq + mult(tapq[2],sync0) - mult(tapq[1],sync1);
			end
			else if(valid3)
			begin

				sumi <=  sumi + mult(tapi[3],sync0) - mult(tapi[4],sync1);
				sumq <=  sumq + mult(tapq[3],sync0) + 12'h800;  //2048 for final round-
				ing
				idata <= ramot[23:12];
				qdata <= ramot[11:0];
			end
			else if(valid4)
			begin
				chani <= chani_[23:12];
				chanq <= chanq_[23:12];
			end
			end
			//intsumi = (chani[11])? {20'hfffff,chani[11:0]}:chani;
			//intsumq = (chanq[11])? {20'hfffff,chanq[11:0]}:chanq;
			//if(chan_val) $display(intsumi*intsumi+intsumq*intsumq);
			/*FOLDENDS*/
		end
		end
		assign chani_ = sumi + mult(tapi[5],sync2) + 12'h800;
		assign chanq_ = sumq + mult(tapq[5],sync2) - mult(tapq[4],sync1);
		assign avchan_ = avchannel + 24'h000800;
		/*FOLDENDS*/
/*FOLDBEGINS 0 2 "Calculate channel"*/
always @(posedge clk)
begin
		if(resynch)
		begin
			chan_val0   <= 1'b0;
			chan_val1   <= 1'b0;
			chan_val2   <= 1'b0;
			chan_val3   <= 1'b0;
			chan_val4   <= 1'b0;
			out_valid   <= 1'b0;
		end
		else
		begin
			chan_val0 <= chan_val;
			chan_val1 <= chan_val0;
			chan_val2 <= chan_val1;
			chan_val3 <= chan_val2;
			chan_val4 <= chan_val3;
			//out_valid <= chan_val4;
			out_valid <= chan_val4&&ramdatavalid&&!pilotdata[1];
		end
		if(chan_val)
			sumsqtemp <= sum[22:11];
			if(chan_val0)
```

```
        topreal <= sum[23:12];
        if(chan_val1)
        topimag <= sum[23:12];
        if(chan_val2)
        sumsq <= sum[23:12];
        if(chan_val3)
        begin

          outitemp <= divider(topreal,sumsq,(constell==0));
          outitem <= divplussoft(topreal,sumsq,constell);
      end
      if(chan_val4)
      begin
          outq <= divider(topimag,sumsq,(constell==0));
          outi <= outitemp;
      end
      //intouti = (outi[7])? {24'hffffff,outi[7:0]}:outi;
      //intoutq = (outq[7])? {24'hffffff,outq[7:0]}:outq;
      //if(chan_val&&ramdatavalid) $display(intsumi);
      //if(chan_val4&&ramdatavalid) $displayb(outitemp,,outitem);
      end
      always @(chan_val or chan_val0 or chan_val1 or chani or chanq or constell
                 or idata or qdata or sumsqtemp)
                 begin
      if(chan_val)
      sum = smult(chani,chani,1) + smult(chanq,chanq,1) + 24'h000400;
      else if(chan_val0)
      sum = smult(idata,chani,1) + smult(qdata,chanq,1) + 24'h000800;
      else if(chan_val1)
      sum = smult(qdata,chani,1) - smult(idata,chanq,1) + 24'h000800;
      else //chan_val2
      begin
        case(constell)
        2'b00:
           sum = smult(sumsqtemp,`SCALEFACTORQPS,0) + 24'h000800;
           2'b01:
           sum = smult(sumsqtemp,`SCALEFACTOR16Q,0) + 24'h000800;
           default:
           sum = smult(sumsqtemp,`SCALEFACTOR64Q,0) + 24'h000800;
           endcase
      end
      end
      /*FOLDENDS*/
  /*FOLDBEGINS 0 2 "Extract Continual and scattered pilots for Freq + Sampling Error
     Block"*/
  always @(posedge clk)
  begin
      if(resynch)
      contloccount <= 6'b0;
      else
      if(ramdatavalid&&valid2&&(pilotaddr==contloc))
         contloccount <= (contloccount == 44)? 6'b0 : contloccount + 1'b1;
         if(ramdatavalid&&valid2&&((pilotaddr==contloc)||pilot))
         uncorrected_iq <= ramot;
         uc_pilots <=
      ramdatavalid&&framedata&&(pilotaddr==contloc)&&valid2&&!resynch;
```

```
        us_pilots <= ramdatavalid&&framedata&&pilot&&valid2&&!resynch;
        u_symbol <= !resynch&&ramdatavalid&&(valid2? (pilotaddr==0) : u_symbol);
        //$display(pilotaddr,,ramot[23:12],,valid2,,contloccount,,uncorrected_iq[
      23:12],,uncorrected_iq[11:0],,uc_pilots,,us_pilots);

    end
    /*FOLDENDS*/
/*FOLDBEGINS 0 2 "Extract TPS pilots "*/
always @(posedge clk)
begin
      if(resynch)
      begin
        tpscount  <= 5'b0;
        tps_pilots <= 3'b0;
        tps_valid <= 1'b0;
        ct_pilots <= 1'b0;
      end
      else
      begin
       if(ramdatavalid&&valid2&&(pilotaddr==tpsloc))
       tpscount <= (tpscount[4])? 5'b0 : tpscount + 1'b1;
       tps_pilots[0] <= valid2? ramdatavalid&&framedata&&(pilotaddr==tpsloc) :
         tps_pilots[0];
       tps_pilots[1] <= (chan_val? tps_pilots[0] : tps_pilots[1]);
       tps_pilots[2] <= tps_pilots[1]&&chan_val3;
       tps_valid <= (tpscount==0)&&tps_pilots[2];
       ct_pilots <= tps_pilots[2];
      end
      if(resynch)
        tpsmajcount <= 6'b0;
        else
        begin
        if(tps_pilots[2])
        begin
          if(tpscount==0)
          begin
                tpsmajcount <= 6'b0;
                out_tps   <= tpsmajcount_[5];
          end
          else
                tpsmajcount <= tpsmajcount_;
                end
      end
      if(resynch)
        pilotdata <= 2'b0;
        else
        begin
       if(valid2)
       pilotdata[0] <= ramdatavalid&&framedata&&(
                                   (pilotaddr==tpsloc)||
                                   (pilotaddr==contloc)||
                                   pilot
                                   );
        pilotdata[1] <= chan_val0? pilotdata[0] : pilotdata[1];
        end
```

```
    //$display(pilotaddr,,ramot[23:12],,valid2,,contloccount,,uncorrected_iq[2
    3:12],,uncorrected_iq[11:0],,uc_pilots,,us_pilots);
//$display(valid2,,pilotdata[0],,pilotdata[1],,pilotdata[2],,ct_pilots,,,,
,,out_valid,,pilotaddr);
  end
  assign tpsmajcount_ = tps(topreal[11],tpscount,tpsmajcount);

  /*FOLDENDS*/
  /*FOLDBEGINS 1 2 "pilot locate control "*/
  always @(posedge clk)
  begin
    if(resynch)
    pilotlocated <= 1'b0;
    else
    if(found_pilots)
    begin
        pilotlocated <= 1'b1;
        pwhichsymbol <= which_symbol + 2'b10;
      end
      end
      /*FOLDENDS*/
/*FOLDBEGINS 0 2 "RAM"*/
always @(posedge clk)
begin
    if(pilotlocated)
    begin
      wrstrb <= !valid0;
      if(valid)
        ramindata <= fftdata;
        pilotaddr <= ramaddr_ - cpoffset;
        ramaddr <= rwtoggle? ramaddr_ : ramaddrrev_;
        if(valid5) ramot <= ramout;
    end
    else
    begin
  /*FOLDBEGINS 0 4 ""*/
  wrstrb <= pilotwrstrb_;
  ramindata <= pilotramin_;
  ramaddr <= pilotramaddr_;
  /*FOLDENDS*/
    end
    ramout <= ramoutdata;
  end
  assign ramaddr_ = (tapinit||framedata&&(valid2&&(count12==11)))? tapcount :
  fftcount;
  assign ramaddrrev_ =
  {ramaddr_[0],ramaddr_[1],ramaddr_[2],ramaddr_[3],ramaddr_[4],ramaddr_[5],

  ramaddr_[6],ramaddr_[7],ramaddr_[8],ramaddr_[9],ramaddr_[10]};
                                      /*FOLDENDS*/
                                      assign c_symbol = whichsymbol[0];

 /*FOLDBEGINS 0 0 ""*/
 always @(posedge clk)
 begin
```

```
 //$display(chan_val,,framedata,,frav,,firstfrav,,,,valid2,,valid4,,out_valid
,,avchannel,,avchan,,sumsqtemp,,,avlow,,chan_val1,,);
//$display(tps_valid,,out_tps,,tpscount,,tps_pilots[2]);
 //$display(in_data,,filtgo,,valid4,,tapload,,,nscat,,count12,,fftcount,,incw
hichsymbol,,,
//tapcount,,ramaddr,,wrstrb,,rwtoggle
//);
 //(resynch,,valid,,fftcount,,ramaddr,,ramindata[23:12],,ramoutdata[23:12],,t
apinit,,tapinit2,,tapcount,,ramout[23:12],,
//tapi[0],,tapi[1],,tapi[2],,tapi[3],,tapi[4],,tapi[5]);
 //$display(tapcount,,tapinit2,,valid4,,valid,,valid2,,wrstrb,,fftcount,,fram
edata,,count12,,tapi[0],,tapi[1],,tapi[2],,tapi[3],,tapi[4],,tapi[5]);
 //$display(,,,,intouti,,intoutq,,out_valid,,,,valid4,,valid2,,chan_val,,filt
go,,framedata,,fftcount,,ramindata[23:12]);
//if(whichsymbol==1)
$display(tapinit,,tapcount,,fftcount,,ramindata[23:12],,,,tapcount,,tapi[0]
,,tapi[1],,tapi[2],,tapi[3],,tapi[4],,tapi[5],,intsumi,,intsumq,,idata,,qda ta);
 //$display(framedata,,pilotaddr,,fftcount,,tapcount,,ramaddr,,ramout[23:12],
,ramindata[23:12],,prbs,,us_pilots,,uc_pilots,,ct_pilots,,out_valid,,,contl occount,,
//tps_pilots[0],,tps_pilots[1],,tps_pilots[2]);
end
/*FOLDENDS*/
pilloc pilloc (.clk(clk), .resync(resync), .in_valid(in_valid), .in_data(in_data),
.found_pilots(found_pilots), .which_symbol(which_symbol),
                        .cpoffset(cpoffset), .incfreq(incfreq),
                        .ramaddr(pilotramaddr_) , .ramin(pilotramin_), .ramout(ramout),
                        .wrstrb(pilotwrstrb_));
/*FOLDBEGINS 0 2 "functions"*/
/*FOLDBEGINS 0 0 "tps demod "*/
function [5:0] tps;
input tpssign;
input [4:0] tpscount;
input [5:0] tpsmajcount;
reg tpsflip;
begin
     case(tpscount)
     5'b00001,5'b00011,5'b00100,5'b00110,5'b01011,5'b01110:
          tpsflip = 0; //added1 since tpscount already incremented
          default:
          tpsflip = 1;
          endcase
          tps = (tpsflip^tpssign)? tpsmajcount - 1'b1 : tpsmajcount + 1'b1;
  end
  endfunction
  /*FOLDENDS*/
  /*FOLDBEGINS 0 0 "pseudo function"*/

  function [11:0] pseudo;
  input [11:0] data;
  input flip;
  begin
     pseudo = flip? ~data + 1'b1 : data;
     end
     endfunction
     /*FOLDENDS*/
     /*FOLDBEGINS 0 0 "averager multiplier"*/
```

```
    function [11:0] avmult;
    input [11:0] i;
    reg [23:0] res;
    begin
    res  = (i*`AVERAGESF) + 23'h000800;  //multiply and round
    avmult = res[23:12];
 end
 endfunction
 /*FOLDENDS*/
 /*FOLDBEGINS 0 0 "filter tap multiplier"*/
 function [27:0] mult;
 input [11:0] i;
 input [11:0] j;
 reg [23:0] res;
 reg [11:0] modi;
 reg [11:0] invi;
 begin
    invi = ~i + 1'b1;
    modi = i[11]? invi : i;
    res = (modi*j);  //multiply and round
    mult = i[11]? {4'hf,~res} + 1'b1 : res;
 end
 endfunction
 /*FOLDENDS*/
 /*FOLDBEGINS 0 0 "signed multiplier"*/
 function [23:0] smult;
 input [11:0] i;
 input [11:0] j;
 input signedj;
 reg [23:0] res;
 reg [11:0] modi;
 reg [11:0] modj;
 begin
    modi = i[11]? ~i + 1'b1 : i;
    modj = (j[11]&&signedj)? ~j + 1'b1 : j;
    res = (modi*modj);
    smult = (i[11]^(j[11]&&signedj))? ~res + 1'b1 : res;
 end
 endfunction
 /*FOLDENDS*/
 /*FOLDBEGINS 0 0 "divider function"*/
 function [7:0] divider;
 input [11:0] dividend;
 input [11:0] divisor;
 input qpsk;

 reg [11:0] moddividend;
 reg signresult;
 reg [12:0] intval;
 reg [12:0] carry;
 reg [7:0] divide;
 reg [8:0] signeddivide;
 integer i;
 begin
    signresult = dividend[11];
    moddividend = dividend[11]? ~dividend + 1'b1 : dividend;
```

```
      divide = 0;
      carry = qpsk? {1'b0,moddividend}:{moddividend,1'b0};
   /*FOLDBEGINS 0 2 ""*/
   for(i=0;i<8;i=i+1)
   begin
         intval = carry - divisor;
         divide[7-i] = !intval[12];
         carry = (intval[12])? {carry[11:0],1'b0} : {intval[11:0],1'b0};
      end
      /*FOLDENDS*/
      //signeddivide = signresult? ~divide + 2'b10 : divide + 1'b1;
      signeddivide = signresult? {1'b1,~divide} + 2'b10 : {1'b0,divide} + 1'b1;
      //$displayb(signeddivide,,divide,,signresult,,constellation,,);
      divider = signeddivide[8:1];
   end
   endfunction
   /*FOLDENDS*/
   /*FOLDBEGINS 0 0 "divider function with soft decisions added"*/
   function [5:0] divplussoft;
   input [11:0] dividend;
   input [11:0] divisor;
   input [1:0] constellation;
   reg [11:0] moddividend;
   reg signresult;
   reg [12:0] intval;
   reg [12:0] carry;
   reg [8:0] divide;
   reg [10:0] signeddivide;
   reg [11:0] fracdivide;
   integer i;
   begin
      signresult = dividend[11];
      moddividend = dividend[11]? ~dividend + 1'b1 : dividend;
      divide = 0;
      carry = (constellation==0)? {1'b0,moddividend}:{moddividend,1'b0};
   /*FOLDBEGINS 0 2 ""*/
   for(i=0;i<9;i=i+1)
   begin
         intval = carry - divisor;
         divide[8-i] = !intval[12];
         carry = (intval[12])? {carry[11:0],1'b0} : {intval[11:0],1'b0};
      end
      /*FOLDENDS*/
      signeddivide = signresult? {2'b11,~divide} + 1'b1 : {2'b0,divide};

      //$displayb(signeddivide,,divide,,signresult,,constellation,,);
   /*FOLDBEGINS 0 2 "qpsk"*/
   if(constellation==2'b0)
   begin
         //$writeh(,,signeddivide,,,,);
         signeddivide = signeddivide + 8'h80;
         //$writeh(signeddivide,,,,);
         if(signeddivide[10])
            fracdivide  = 9'h0;
            else
            if(signeddivide[9]||signeddivide[8])
```

```
              fracdivide  = 12'h700;
              else
              begin
              fracdivide = signeddivide[7:0] + {signeddivide[7:0],1'b0} +
              {signeddivide[7:0],2'b0}; //*7
              fracdivide = fracdivide + 8'h80;
            end
            divplussoft = {3'b0,fracdivide[10:8]};
          end
          else
          /*FOLDENDS*/
        /*FOLDBEGINS 0 2 "16qam"*/
        if(constellation==2'b01)
        begin
            $writeh(,,signeddivide,,,,);
            signeddivide = signeddivide + 8'hc0;
            $writeh(,,signeddivide,,,,);
            if(signeddivide[10])
            begin
              signeddivide = 10'b0;
              fracdivide  = 9'h0;
            end
            else
            if(signeddivide[9]||(signeddivide[8:7]==2'b11))
            begin
              fracdivide  = 12'h380;
              signeddivide = 10'h100;
            end
            else
            begin
              fracdivide = signeddivide[6:0] + {signeddivide[6:0],1'b0} +
              {signeddivide[6:0],2'b0}; //*7
              fracdivide = fracdivide + 8'h40;
            end
            divplussoft = {1'b0,signeddivide[8:7],fracdivide[9:7]};
          end
          /*FOLDENDS*/
        /*FOLDBEGINS 0 2 "32qam"*/
        else
        begin
            signeddivide = signeddivide + 8'he0;
            if(signeddivide[10])
            begin

              signeddivide = 10'b0;
              fracdivide  = 9'h0;
            end
            else
            if(signeddivide[9]||(signeddivide[8:6]==3'b111))
            begin
              signeddivide = 10'h180;
              fracdivide  = 9'h1c0;
            end
            else
            begin
```

```
            fracdivide = signeddivide[5:0] + {signeddivide[5:0],1'b0} +
            {signeddivide[5:0],2'b0}; //*7
            fracdivide = fracdivide + 8'h20;
        end
        divplussoft = {signeddivide[8:6],fracdivide[8:6]};
    end
    /*FOLDENDS*/
end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "PRBS alpha3/6/9/12 multiplier"*/
function [10:0] alpha;
input [1:0] which_symbol;
begin
    case(which_symbol)
    2'b0:
    alpha = 11'b11111111111;
    2'b01:
    alpha = 11'b00011111111;
    2'b10:
    alpha = 11'b00000011111;
    2'b11:
    alpha = 11'b00000000011;
    endcase
end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "PRBS alpha12 multiplier"*/
function [10:0] alpha12;
input [10:0] prbsin;
reg [10:0] prbs0;
reg [10:0] prbs1;
reg [10:0] prbs2;
reg [10:0] prbs3;
reg [10:0] prbs4;
reg [10:0] prbs5;
reg [10:0] prbs6;
reg [10:0] prbs7;
reg [10:0] prbs8;
reg [10:0] prbs9;
reg [10:0] prbs10;
begin
    prbs0  = {prbsin[0] ^ prbsin[2],prbsin[10:1]};

    prbs1  = {prbs0[0]  ^ prbs0[2] ,prbs0[10:1]};
    prbs2  = {prbs1[0]  ^ prbs1[2] ,prbs1[10:1]};
    prbs3  = {prbs2[0]  ^ prbs2[2] ,prbs2[10:1]};
    prbs4  = {prbs3[0]  ^ prbs3[2] ,prbs3[10:1]};
    prbs5  = {prbs4[0]  ^ prbs4[2] ,prbs4[10:1]};
    prbs6  = {prbs5[0]  ^ prbs5[2] ,prbs5[10:1]};
    prbs7  = {prbs6[0]  ^ prbs6[2] ,prbs6[10:1]};
    prbs8  = {prbs7[0]  ^ prbs7[2] ,prbs7[10:1]};
    prbs9  = {prbs8[0]  ^ prbs8[2] ,prbs8[10:1]};
    prbs10 = {prbs9[0]  ^ prbs9[2] ,prbs9[10:1]};
    alpha12 = {prbs10[0] ^ prbs10[2],prbs10[10:1]};
end
```

```
   endfunction
   /*FOLDENDS*/
   /*FOLDENDS*/
endmodule
```

Listing 19

```
/*FOLDBEGINS 0 0 "Copyright"*/
/*********************************************************
Copyright (c) Pioneer Digital Design Centre Limited

NAME:  pilloc_rtl.v

PURPOSE: Pilot location

CREATED:    June 1997  BY: J. Parker (C code)

MODIFIED:         BY: T. Foxcroft

USED IN PROJECTS:  cofdm only.

*********************************************************/
/*FOLDENDS*/
`define FFTSIZE  2048
`define SCATNUM  45
module pilloc (clk, resync, in_valid, in_data, found_pilots, which_symbol, cpoffset,
incfreq,
                        ramaddr , ramin, ramout, wrstrb);
                        /*FOLDBEGINS 0 0 "i/o"*/
                        input clk, resync, in_valid;
                        input [23:0] in_data;
                        output found_pilots;
                        output [1:0] which_symbol;
                        output [10:0] cpoffset;
                        output incfreq;
                        /*FOLDENDS*/

                        /*FOLDBEGINS 0 0 "ram i/o"*/
                        output [10:0] ramaddr;
                        reg   [10:0] ramaddr_;
                        output [23:0] ramin;
                        input  [23:0] ramout;
                        output wrstrb;
                        reg [10:0] ramaddr;
                        reg [23:0] ramin;
                        reg wrstrb;
                        /*FOLDENDS*/
                        /*FOLDBEGINS 0 0 "vars"*/
                        reg found_pilots;
                        reg [1:0] which_symbol;
                        reg [1:0] which_symbolcount;
                        reg [1:0] which_symbol_;
                        reg [10:0] cpoffset;
                        reg incfreq;
```

```
                    reg found_pilot;
                    reg [19:0] v;
                    reg [19:0] sum;
                    reg [3:0] splocoffset;
                    wire [10:0] carrier_number;
                    reg [10:0] continual_pilot_offset;

reg resynch;
reg [3:0] valid;
reg [23:0] fftdata;
reg [10:0] fftcount;
reg contcomplete;
reg firstcontsearch;
reg finishedsearch;
reg [4:0] firstscatcomplete;
reg [4:0] failedtolock;
reg [2:0] spmax;
reg [2:0] spmaxfirst;
reg [10:0] pilot_offset;
reg [1:0] sploc1zero;
reg [10:0] sploc0;
reg [5:0] sploc1;
reg [10:0] splocmaxcount;

reg [3:0] spoffset;
reg [19:0] sumscat [11:0];
reg [19:0] sumscatmax;
reg [3:0] sumscatmaxno0;
reg [3:0] sumscatmaxno1;
wire [19:0] sumscat1;
wire [19:0] sumscat3;
wire [19:0] sumscat5;
reg [11:0] sumscatfirst;
reg [4:0] fftfinished;
reg ramwritestop; //botch for development purposes
wire [3:0] mod12fftcount;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "continuous pilot location"*/
reg [10:0] contloc;
always @(sploc1)
begin
   case(sploc1)
   6'b000000: contloc = 0;
   6'b000001: contloc = 48;
   6'b000010: contloc = 54;
   6'b000011: contloc = 87;
   6'b000100: contloc = 141;
   6'b000101: contloc = 156;
   6'b000110: contloc = 192;
   6'b000111: contloc = 201;
   6'b001000: contloc = 255;
   6'b001001: contloc = 279;
   6'b001010: contloc = 282;
   6'b001011: contloc = 333;
   6'b001100: contloc = 432;
   6'b001101: contloc = 450;
```

```
    6'b001110: contloc = 483;
    6'b001111: contloc = 525;
    6'b010000: contloc = 531;
    6'b010001: contloc = 618;
    6'b010010: contloc = 636;
    6'b010011: contloc = 714;
    6'b010100: contloc = 759;
    6'b010101: contloc = 765;
    6'b010110: contloc = 780;
    6'b010111: contloc = 804;
    6'b011000: contloc = 873;
    6'b011001: contloc = 888;
    6'b011010: contloc = 918;
    6'b011011: contloc = 939;
    6'b011100: contloc = 942;
    6'b011101: contloc = 969;
    6'b011110: contloc = 984;
    6'b011111: contloc = 1050;
    6'b100000: contloc = 1101;
    6'b100001: contloc = 1107;
    6'b100010: contloc = 1110;
    6'b100011: contloc = 1137;
    6'b100100: contloc = 1140;
    6'b100101: contloc = 1146;
    6'b100110: contloc = 1206;
    6'b100111: contloc = 1269;
    6'b101000: contloc = 1323;
    6'b101001: contloc = 1377;
    6'b101010: contloc = 1491;
    6'b101011: contloc = 1683;
    default:  contloc = 1704;
    endcase
end
/*FOLDENDS*/

always @(posedge clk)
begin
    resynch <= resync;
    if(resynch)
    begin
        valid       <= 4'b0;
        fftcount     <= 11'b0;
        firstscatcomplete  <= 5'b0;
        sum         <= 20'b0;
        sploc0      <= 11'b0;
        sploc1      <= 6'b0;
        contcomplete    <= 1'b0;
        failedtolock    <= 5'b0;
        spmax       <= 1'b0;
        spmaxfirst    <= 1'b0;
        ramwritestop    <= 1'b0;
        found_pilots    <= 1'b0;
        found_pilot    <= 1'b0;
        firstcontsearch    <= 1'b0;
        finishedsearch    <= 1'b0;
        which_symbolcount  <= 2'b0;
```

```
            incfreq      <= 1'b0;
        end
        else
        begin
            incfreq   <= !failedtolock[1]&&failedtolock[0]&&fftfinished[4];
            found_pilots <= !found_pilot&&finishedsearch;
            found_pilot <= finishedsearch;
            valid[0] <= in_valid;
            valid[1] <= valid[0];
            valid[2] <= valid[1];
            valid[3] <= valid[2];
            fftdata <= in_data;
            if(valid[0]&&!finishedsearch)
                fftcount <= fftcount + 1'b1;
                //if(fftfinished[0])
                // $display("frame",,fftcount);
                //if(incfreq)
                // $display("tweek");

    /*FOLDBEGINS 0 4 "locate continual pilots"*/
    spmax[1]    <= spmax[0];
    spmax[2]    <= spmax[1];
    spmaxfirst[1] <= spmaxfirst[0];
    spmaxfirst[2] <= spmaxfirst[1];
    //if(fftfinished[3])
    // $display(spoffset,,which_symbol);

            if(fftfinished[3])
            begin
                failedtolock[1] <= failedtolock[0];
                failedtolock[2] <= failedtolock[1];
                failedtolock[3] <= failedtolock[2];
                failedtolock[4] <= failedtolock[3];

                if(failedtolock[0])
                begin
            /*FOLDBEGINS 0 2 ""*/
            if(failedtolock[4])
                        failedtolock[0] <= 1'b0;
                        firstscatcomplete  <= 5'b0;
                        ramwritestop    <= 1'b0;
                        firstcontsearch   <= 1'b0;
                /*FOLDENDS*/
                end
                else
                begin
            /*FOLDBEGINS 0 4 ""*/
            firstscatcomplete[0] <= 1'b1;
            firstcontsearch   <= !firstscatcomplete[0];
            ramwritestop <= !ramwritestop||finishedsearch;
            contcomplete <= ramwritestop;
            if(!finishedsearch&&firstscatcomplete[0]&&ramwritestop)
            begin
                        finishedsearch  <= firstcontsearch? 1'b0 :
                        (cpoffset==continual_pilot_offset);
                        cpoffset     <= continual_pilot_offset;
```

```
failedtolock[0] <= !firstcontsearch&&(cpoffset!=continual_pilot_offset);
end
/*FOLDENDS*/
end
end
else
begin
firstscatcomplete[1] <= firstscatcomplete[0]&&!contcomplete;
firstscatcomplete[2] <= firstscatcomplete[1];
if(firstscatcomplete[0]&&!finishedsearch&&!contcomplete&&!finishedsearc h
    &&(sploc1==44)&&(sploc0==splocmaxcount))
    contcomplete  <= 1'b1;
end
if(found_pilots)
    $display(which_symbol,,cpoffset,,spoffset);
    //$display(sum,,contcomplete,,ramwritestop,,which_symbol,,spoffset,,,splo
c0,,splocmaxcount,,v,,,,,,fftfinished[3],,finishedsearch);
    //$display(fftcount,,firstscatcomplete[0],,ramwritestop,,spoffset,,sumsca
tmaxno1,,,,finishedsearch,,found_pilots,,
    //,,,,,,,,
    //pilot_offset,,which_symbol,,,,cpoffset,,failedtolock );
    sploc1zero[0]  <= (sploc1 == 0);
    sploc1zero[1]  <= sploc1zero[0];

if(firstscatcomplete[0]&&!finishedsearch&&!contcomplete&&!finishedsearch)
    begin
  if(sploc1==44)

  begin
/*FOLDBEGINS 0 4 ""*/
//$display(sploc0,,splocmaxcount);
pilot_offset <= sploc0 + splocoffset;
which_symbol <= which_symbol_ - which_symbolcount;
if(sploc0==splocmaxcount)
begin
        sploc0     <= 11'b0;
        //contcomplete   <= 1'b1;
        which_symbolcount <= 2'b0;
    end
    else
    begin
        sploc0 <= sploc0 + 2'b11;
        which_symbolcount <= which_symbolcount + 1'b1;
    end
    if(sploc0==0)
        spmaxfirst[0] <= 1'b1;
        sploc1 <= 6'b0;
        spmax[0] <= 1'b1;
        /*FOLDENDS*/
  end
  else
  begin
/*FOLDBEGINS 0 4 ""*/
sploc1 <= sploc1 + 1'b1;
spmax[0] <= 1'b0;
spmaxfirst[0] <= 1'b0;
```

```
      /*FOLDENDS*/
        end
        end
        if(firstscatcomplete[2])
        begin
        if(sploc1zero[1])
        sum <= modulus(ramout[23:12],ramout[11:0]);
        else
        sum <= modulus(ramout[23:12],ramout[11:0]) + sum;
      end
      /*FOLDENDS*/
    end
    /*FOLDBEGINS 0 2 "search for largest continous pilot correlation"*/
    if(spmax[2])
    begin
      if(spmaxfirst[2])
      begin
        v <= sum;
        continual_pilot_offset <= pilot_offset;
      end
      else
      begin
        if(sum>v)
        begin
          v <= sum;
          continual_pilot_offset <= pilot_offset;

        end
        end
        //$display(sum,,continual_pilot_offset,,contcomplete,,ramwritestop,,which
      _symbol,,spoffset,,,sploc0,,splocmaxcount,,v);
        //$display(sum);
    end
    /*FOLDENDS*/
  end
  assign carrier_number = contloc + sploc0 + splocoffset;
  /*FOLDBEGINS 0 0 "scattered pilot offset mod 3"*/
  always @(spoffset)
  begin
    splocoffset = 2'b0;
    splocmaxcount = 342;
    which_symbol_ = 2'b0;
    case(spoffset)
      4'b0000,4'b0011,4'b0110,4'b1001:
      begin
        splocoffset = 2'b0;
        splocmaxcount = 342;
      end
      4'b0001,4'b0100,4'b0111,4'b1010:
      begin
        splocoffset = 2'b01;
        splocmaxcount = 339;
      end
      //4'b0010,4'b0101,4'b1000,4'b1011:
      default:
      begin
```

```
splocoffset = 2'b10;
splocmaxcount = 339;
end
endcase
case(spoffset)
4'b0000,4'b0001,4'b0010:
which_symbol_ = 2'b0;
4'b0011,4'b0100,4'b0101:
which_symbol_ = 2'b01;
4'b0110,4'b0111,4'b1000:
which_symbol_ = 2'b10;
//4'b1001,4'b1010,4'b1011:
default:
which_symbol_ = 2'b11;
endcase
end
/*FOLDENDS*/
/*FOLDBEGINS 1 0 "Search for scattered pilots"*/
always @(posedge clk)
begin
if(resynch)
sumscatfirst <= 12'hfff;
else

begin
if(valid[0]&&!finishedsearch)
/*FOLDBEGINS 1 2 "do the accumulations"*/
case(mod12fftcount)
4'h0:
begin
sumscat[0] <= (sumscatfirst[0])? modulus(fftdata[23:12],fftdata[11:0]) :
sumscat[0] + modulus(fftdata[23:12],fftdata[11:0]);
sumscatfirst[0] <= 1'b0;
end
4'h1:
begin
sumscat[1] <= (sumscatfirst[1])? modulus(fftdata[23:12],fftdata[11:0]) :
sumscat[1] + modulus(fftdata[23:12],fftdata[11:0]);
sumscatfirst[1] <= 1'b0;
end
4'h2:
begin
sumscat[2] <= (sumscatfirst[2])? modulus(fftdata[23:12],fftdata[11:0]) :
sumscat[2] + modulus(fftdata[23:12],fftdata[11:0]);
sumscatfirst[2] <= 1'b0;
end
4'h3:
begin
sumscat[3] <= (sumscatfirst[3])? modulus(fftdata[23:12],fftdata[11:0]) :
sumscat[3] + modulus(fftdata[23:12],fftdata[11:0]);
sumscatfirst[3] <= 1'b0;
end
4'h4:
begin
```

```
        sumscat[4] <= (sumscatfirst[4])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[4] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[4] <= 1'b0;
    end
    4'h5:
    begin
        sumscat[5] <= (sumscatfirst[5])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[5] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[5] <= 1'b0;
    end
    4'h6:
    begin
        sumscat[6] <= (sumscatfirst[6])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[6] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[6] <= 1'b0;
    end
    4'h7:
    begin
        sumscat[7] <= (sumscatfirst[7])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[7] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[7] <= 1'b0;
    end
    4'h8:
    begin

        sumscat[8] <= (sumscatfirst[8])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[8] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[8] <= 1'b0;
    end
    4'h9:
    begin
        sumscat[9] <= (sumscatfirst[9])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[9] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[9] <= 1'b0;
    end
    4'ha:
    begin
        sumscat[10] <= (sumscatfirst[10])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[10] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[10] <= 1'b0;
    end
    default:
    begin
        sumscat[11] <= (sumscatfirst[11])? modulus(fftdata[23:12],fftdata[11:0]) :
        sumscat[11] + modulus(fftdata[23:12],fftdata[11:0]);
        sumscatfirst[11] <= 1'b0;
    end
    endcase
    /*FOLDENDS*/
    else if(fftfinished[0])
        sumscatfirst <= 12'hfff;
        end
 /*FOLDBEGINS 1 0 "Find offset"*/
 if(resynch)
        fftfinished <= 5'b0;
        else
```

```
        begin
        fftfinished[0] <= valid[0]&&!finishedsearch&&(fftcount==2047);
        fftfinished[1] <= fftfinished[0];
        fftfinished[2] <= fftfinished[1];
        fftfinished[3] <= fftfinished[2];
        fftfinished[4] <= fftfinished[3];
     end
     if(!ramwritestop)
     begin
        if(fftfinished[0])
        begin
           sumscat[0] <= (sumscat[0] > sumscat[1])? sumscat[0] : sumscat[1];
           sumscat[1] <= (sumscat[0] > sumscat[1])? 0 : 1;
           sumscat[2] <= (sumscat[2] > sumscat[3])? sumscat[2] : sumscat[3];
           sumscat[3] <= (sumscat[2] > sumscat[3])? 2 : 3;
           sumscat[4] <= (sumscat[4] > sumscat[5])? sumscat[4] : sumscat[5];
           sumscat[5] <= (sumscat[4] > sumscat[5])? 4 : 5;
           sumscat[6] <= (sumscat[6] > sumscat[7])? sumscat[6] : sumscat[7];
           sumscat[7] <= (sumscat[6] > sumscat[7])? 6 : 7;
           sumscat[8] <= (sumscat[8] > sumscat[9])? sumscat[8] : sumscat[9];
           sumscat[9] <= (sumscat[8] > sumscat[9])? 8 : 9;
           sumscat[10] <= (sumscat[10]>sumscat[11])? sumscat[10] : sumscat[11];
           sumscat[11] <= (sumscat[10]>sumscat[11])? 10 : 11;

        end
        if(fftfinished[1])
        begin
           sumscat[0] <= (sumscat[0] > sumscat[2])? sumscat[0] : sumscat[2];
           sumscat[1] <= (sumscat[0] > sumscat[2])? sumscat[1] : sumscat[3];
           sumscat[2] <= (sumscat[4] > sumscat[6])? sumscat[4] : sumscat[6];
           sumscat[3] <= (sumscat[4] > sumscat[6])? sumscat[5] : sumscat[7];
           sumscat[4] <= (sumscat[8] > sumscat[10])? sumscat[8] : sumscat[10];
           sumscat[5] <= (sumscat[8] > sumscat[10])? sumscat[9] : sumscat[11];
        end
        if(fftfinished[2]&&!ramwritestop)
           spoffset <= sumscatmaxno1;
           end
           if(fftfinished[0])
           begin
        $display(sumscat[0]);
        $display(sumscat[1]);
        $display(sumscat[2]);
        $display(sumscat[3]);
        $display(sumscat[4]);
        $display(sumscat[5]);
        $display(sumscat[6]);
        $display(sumscat[7]);
        $display(sumscat[8]);
        $display(sumscat[9]);
        $display(sumscat[10]);
        $display(sumscat[11]);
        $display();
      end

   end
```

```
always @(sumscat[0] or sumscat[1] or sumscat[2] or sumscat[3] or sumscat[4] or
sumscat[5]
            or sumscat1 or sumscat3 or sumscat5)
            begin
  sumscatmax   = (sumscat[0] > sumscat[2])? sumscat[0]  : sumscat[2];
  sumscatmaxno0  = (sumscat[0] > sumscat[2])? sumscat1[3:0] : sumscat3[3:0];
  sumscatmaxno1  = (sumscatmax > sumscat[4])? sumscatmaxno0 : sumscat5[3:0];
end
assign mod12fftcount = mod12(fftcount);
assign sumscat1 = sumscat[1];
assign sumscat3 = sumscat[3];
assign sumscat5 = sumscat[5];

/*FOLDENDS*/
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "ram"*/
always @(posedge clk)
   ramaddr_ <= ramaddr;
   always @(ramwritestop or valid or finishedsearch or fftcount or carrier_number or
ramwritestop or ramaddr_ or fftdata)
   begin

   ramaddr = ramaddr_;
   if(!ramwritestop)
   begin
      if(valid[0]&&!finishedsearch)
      ramaddr = {fftcount[0],fftcount[1],fftcount[2],fftcount[3],fftcount[4],fftcount[
         5],fftcount[6],
                  fftcount[7],fftcount[8],fftcount[9],fftcount[10]};
                  end
                  else
      ramaddr = carrier_number;
      ramin = fftdata;
      wrstrb = !(!ramwritestop&&valid[1]);
end
/*FOLDENDS*/

/*FOLDBEGINS 0 0 "modulus approximation function"*/
function [11:0] modulus;
input [11:0] i;
input [11:0] j;
reg [11:0] modi;
reg [11:0] modj;
begin
   modi = (i[11]? ~i : i) + i[11];
   modj = (j[11]? ~j : j) + j[11];
   modulus = modi + modj;
end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "mod12"*/
function [3:0] mod12;
input [10:0] count;
reg [14:0] onetwelfth;
reg [7:0] modulus12;
parameter TWELFTH = 12'haab;
```

```
begin
   onetwelfth  = {count[0],count[1],count[2],count[3],count[4],count[5],count [6],
   count[7],count[8],count[9],count[10]} * TWELFTH;
   modulus12 = {onetwelfth[14:9],1'b0} + onetwelfth[14:9] + 4'h8;   //*12
   mod12   = modulus12[7:4];
end
/*FOLDENDS*/
endfunction
endmodule
```

Listing 20

```
// SccsId: @(#)bch_decode.v      1.2 8/22/97
/*FOLDBEGINS 0 0 "copyright"*/
//***********************************************************
// Copyright (c) 1997 Pioneer Digital Design Centre Limited
//
// NAME:  BCH_rtl.v
//
// PURPOSE: BCH decoder for TPS pilots. Flags up to two error
//    positions using search technique.
//
//***********************************************************
/*FOLDENDS*/
`define DATA0_SIZE  7'b0110100
`define DATA1_SIZE  7'b0110111

module bch_decode (clk, resync, in_data, in_valid, in_finalwrite, out_valid, out_data);
/*FOLDBEGINS 0 0 "I/Os"*/
input  clk, resync;
input  in_data, in_valid, in_finalwrite;
output out_valid;
output out_data;
reg   out_data;
reg   out_valid;
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "variables"*/
reg resynch;
reg valid;
reg finalwrite;
reg indata;
reg [6:0] S0;
reg [6:0] S1;
reg [6:0] S2;
reg [6:0] count;

reg search1error, found2error, oneerror, twoerror;
wire twoerror_;
reg noerrors;
reg delay0, delay1, delay2;
reg [6:0] Gs0;
reg [6:0] Gs1;
reg [6:0] Gs2;
/*FOLDENDS*/
always @(posedge clk)
begin
```

```
/*FOLDBEGINS 0 2 "read in data and calculate syndromes"*/
 resynch <= resync;
 if(resynch)
 begin
  valid    <= 1'b0;
  S0     <= 7'b0;
  S1     <= 7'b0;
  S2     <= 7'b0;
 end
 else
 begin
  valid <= in_valid;
  if(delay1&&twoerror_)
  begin
  /*FOLDBEGINS 0 4 "update after one in two errors found"*/
   S0 <= S0^Gs0;
   S1 <= S1^Gs1;
   S2 <= S2^Gs2;
      /*FOLDENDS*/
      end
      else if(valid)
      begin
   S0 <= indata ^ MULTA1(S0);
   S1 <= indata ^ MULTA2(S1);
   S2 <= indata ^ MULTA3(S2);
  end
  end
  indata <= in_data;
  /*FOLDENDS*/
  /*FOLDBEGINS 0 2 "out_valid control"*/
 if(resynch)

 begin
  delay0   <= 1'b0;
  delay1   <= 1'b0;
  delay2   <= 1'b0;
  out_valid  <= 1'b0;
  finalwrite  <= 1'b0;
 end
 else
 begin
  finalwrite <= in_finalwrite;
  if(valid&&finalwrite)
   delay0  <= 1'b1;
   else
   if(count == `DATA1_SIZE-4)
   delay0  <= 1'b0;
  delay1   <= delay0;
  delay2   <= delay1;
  out_valid  <= delay2;
 end
 /*FOLDENDS*/
/*FOLDBEGINS 0 2 "error search algorithm"*/
if(delay0&&!delay1)
begin
  noerrors   <= (S0 == 7'b0);
```

```
    search1error <= (GFULL(S0,S1) == S2);
    found2error <= 1'b0;
    twoerror  <= 1'b0;
    count <= 7'b0;
    Gs0 <= 7'h50;
    Gs1 <= 7'h20;
    Gs2 <= 7'h3d;
  end
  else
  if(delay1)
  begin
    oneerror   <= ((S0^Gs0) == 7'b0)&&search1error;
    twoerror   <= twoerror_;
    if(twoerror_)
    begin
      search1error <= 1'b1;
      found2error <= 1'b1;
    end
    Gs0 <= DIV1(Gs0);
    Gs1 <= DIV2(Gs1);
    Gs2 <= DIV3(Gs2);
    count <= count + 1'b1;
  end
  out_data <= (twoerror||oneerror)&&!noerrors;
   /*FOLDENDS*/
   end
   assign twoerror_ = ( GFULL((S0^Gs0),(S1^Gs1)) ==
(S2^Gs2))&&!found2error&&!twoerror;
   /*FOLDBEGINS 0 0 "functions"*/
   /*FOLDBEGINS 0 0 "GFULL function"*/

   function [6:0] GFULL;
  input [6:0] X;
  input [6:0] Y;
  reg [6:0] A0, A1, A2, A3, A4, A5, A6;
  integer i;
  begin
    A0 = X;
    A1 = {A0[5],A0[4],A0[3],A0[2] ^ A0[6],A0[1],A0[0],A0[6]};
    A2 = {A1[5],A1[4],A1[3],A1[2] ^ A1[6],A1[1],A1[0],A1[6]};
    A3 = {A2[5],A2[4],A2[3],A2[2] ^ A2[6],A2[1],A2[0],A2[6]};
    A4 = {A3[5],A3[4],A3[3],A3[2] ^ A3[6],A3[1],A3[0],A3[6]};
    A5 = {A4[5],A4[4],A4[3],A4[2] ^ A4[6],A4[1],A4[0],A4[6]};
    A6 = {A5[5],A5[4],A5[3],A5[2] ^ A5[6],A5[1],A5[0],A5[6]};

    for(i=0;i<7;i=i+1)
    begin
      A0[i] = A0[i] && Y[0];
      A1[i] = A1[i] && Y[1];
      A2[i] = A2[i] && Y[2];
      A3[i] = A3[i] && Y[3];
      A4[i] = A4[i] && Y[4];
      A5[i] = A5[i] && Y[5];
      A6[i] = A6[i] && Y[6];
    end
    GFULL = A0 ^ A1 ^ A2 ^ A3 ^ A4 ^ A5 ^ A6;
```

```
end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "MULTA1 function"*/
function [6:0] MULTA1;
input [6:0] X;
begin
  MULTA1 = {X[5],X[4],X[3],X[2] ^ X[6],X[1],X[0],X[6]};
  end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "MULTA2 function"*/
function [6:0] MULTA2;
  input [6:0] X;
  begin
    MULTA2 = {X[4],X[3],X[2]^X[6],X[1]^X[5],X[0],X[6],X[5]};
    end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "MULTA3 function"*/
function [6:0] MULTA3;
  input [6:0] X;
  begin
    MULTA3 = {X[3],X[2]^X[6],X[1]^X[5],X[0]^X[4],X[6],X[5],X[4]};
    end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "DIV1 function"*/
function [6:0] DIV1;
  input [6:0] X;
  begin
    DIV1 = {X[0],X[6],X[5],X[4],X[3]^X[0],X[2],X[1]};
    end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "DIV2 function"*/
function [6:0] DIV2;
  input [6:0] X;
  begin
    DIV2 = {X[1],X[0],X[6],X[5],X[4]^X[1],X[3]^X[0],X[2]};
    end
endfunction
/*FOLDENDS*/
/*FOLDBEGINS 0 0 "DIV3 function"*/
function [6:0] DIV3;
  input [6:0] X;
  begin
    DIV3 = {X[2],X[1],X[0],X[6],X[5]^X[2],X[4]^X[1],X[3]^X[0]};
    end
endfunction
/*FOLDENDS*/
/*FOLDENDS*/
/*FOLDBEGINS 0 0 ""*/
//always @(posedge clk)
// $display(in_valid,,in_data,,in_finalwrite,,,,out_valid,,out_data,,,S0,,S1,,S2 ,,,);
//always @(psedge clk)
```

```
// $display(resynch,,in_valid,,in_data,,out_valid,,S0,,S1,,,,count,,,delay0,,del
ay1,,delay2,,,,
// ,,,,delay2,,noerrors,,oneerror,,twoerror,,out_data,,out_valid);
//always @(posedge clk)
// $display(in_valid,,in_data,,,,out_valid,,out_data,,,S0,,S1,,S2,,,);
//always @(posedge clk)
// $display(in_valid,,in_data,,,,out_valid,,out_data,,,S0,,S1,,S2,,,);
/*FOLDENDS*/
endmodule
```

Listing 21

```
// SccsId: @(#)tps.v    1.2 9/15/97
/*FOLDBEGINS 0 0 "copyright"*/
//***************************************************************
// Copyright (c) 1997 Pioneer Digital Design Centre Limited
//
// NAME: tps_rtl.v
//
// PURPOSE: Demodulates TPS pilots using DPSK. Finds sync bits.
//    Corrects up to two errors using BCH.
//   (DPSK produces two errors for each transmission error)
// HISTORY:
// 15/9/97 PK Added scan IO ports, te, tdin ,tdout
//
//***************************************************************
/*FOLDENDS*/
`define SYNCSEQ0 16'b0111011110101100
`define SYNCSEQ1 16'b1000100001010011
module tps (resync, clk, tps_valid, tps_pilot, tps_sync, tps_data, upsel, upaddr,
uprstr, lupdata,
            te, tdin, tdout);
            /*FOLDBEGINS 0 0 "i/os"*/
            input resync, clk, tps_valid, tps_pilot, upsel, uprstr, te, tdin;
            input [1:0] upaddr;
            inout [7:0] lupdata;
            output tps_sync, tdout;
            output [30:0] tps_data;
            /*FOLDENDS*/
            /*FOLDBEGINS 0 0 "registers"*/
            reg resynch;
            reg [1:0] foundsync;
            reg [66:0] tpsreg;
            reg [15:0] syncreg;
            reg [1:0] tpsvalid;
            reg [1:0] pilot;
            reg tps_sync;
            reg [7:0] bch_count;
            reg [2:0] bch_go;
            reg bch_finalwrite;
            wire bch_data;
            wire bch_valid;
            wire bch_error;
            integer i;
            wire upsel0;
            wire upsel1;
```

```
            wire upsel2;
            wire upsel3;
            /*FOLDENDS*/

always @(posedge clk)
begin
/*FOLDBEGINS 0 2 "Synchronise to TPS"*/
   resynch <= resync;
   if(tpsvalid[0]&&!(foundsync[0]||foundsync[1]||tps_sync))
   begin
      tpsreg[66] <= pilot[1]^pilot[0];
      for(i=0;i<66;i=i+1)
         tpsreg[i] <= tpsreg[i+1];

         end
         else
      if(bch_valid&&bch_error)
      tpsreg[bch_count] <= !tpsreg[bch_count];
   if(tpsvalid[0]&&(foundsync[0]||foundsync[1]))
   begin
      syncreg[15] <= pilot[1]^pilot[0];
      for(i=0;i<15;i=i+1)
         syncreg[i] <= syncreg[i+1];
         end

   pilot[0]  <= tps_pilot;
   pilot[1]  <= pilot[0];
   if(resynch)
   begin
      tpsvalid    <= 2'b0;
      tps_sync    <= 1'b0;
      bch_go      <= 3'b0;
      bch_finalwrite <= 1'b0;
      bch_count   <= 8'b0;
      foundsync   <= 2'b0;
   end
   else
   begin
      tpsvalid[0] <= tps_valid;
      tpsvalid[1] <= tpsvalid[0];
      bch_go[1]  <= bch_go[0];
      bch_go[2]  <= bch_go[1];
      bch_finalwrite <= (bch_count == 65)&&bch_go[2];
      if((bch_count == 52)&&bch_valid)
         tps_sync <= 1'b1;
         /*FOLDBEGINS 0 2 "counter"*/
      if(bch_count == 66)
      bch_count <= 8'b0;
      else if(tpsvalid[1]&&!(foundsync[0] || foundsync[1]))
      begin
         if(tpsreg[15:0] == `SYNCSEQ1)
         bch_count <= 8'hfe;      //-2
         if(tpsreg[15:0] == `SYNCSEQ0)
         bch_count <= 8'hfe;      //-2
      end
      else if(tpsvalid[1]&&(bch_count==15)&&(foundsync[0] || foundsync[1]))
```

```
            bch_count <= 8'hfe;      //-2
            else
            begin
        if(bch_valid || bch_go[0] || ((foundsync[0] || foundsync[1])&&tpsvalid[0]))
        bch_count <= bch_count + 1'b1;
      end
      /*FOLDENDS*/
    /*FOLDBEGINS 0 2 "BCH + second SYNC reg control"*/
    if(bch_count == 66)
    begin
        bch_go <= 3'b0;

        end
        else if(tpsvalid[1])
        begin
        if(foundsync[0] || foundsync[1])
        begin
          if(bch_count==15)
          begin
            if(((syncreg[15:0] == `SYNCSEQ0)&&foundsync[1])|| ((syncreg[15:0]
              == `SYNCSEQ1)&&foundsync[0]) )
            bch_go[0] <= 1'b1;
            else
            foundsync <= 2'b0;
          end
          end
          else
          begin
          if(tpsreg[15:0] == `SYNCSEQ1)
          foundsync[1] <= 1'b1;
          if(tpsreg[15:0] == `SYNCSEQ0)
          foundsync[0] <= 1'b1;
        end
        end
        /*FOLDENDS*/
    end
    /*FOLDENDS*/
  end
  assign bch_data = tpsreg[bch_count];
  /*FOLDBEGINS 0 0 ""*/
  //always @(posedge clk)
  //begin
  // $write(tps_valid,,tps_sync,,tps_pilot,,tpsvalid[1],,pilot,,,,,,
  // bch_finalwrite,,,,,,bch_go[2],,bch_data,,bch_valid,,bch_error,,bch_count,,tps
  _sync,,,,,);
  // $displayb(tpsreg,,syncreg,,foundsync);
  //end
  /*FOLDENDS*/
  /*FOLDBEGINS 0 0 "micro access"*/
  assign upsel0  = upsel&&uprstr&&!upaddr[1]&&!upaddr[0];
  assign upsel1  = upsel&&uprstr&&!upaddr[1]&& upaddr[0];
  assign upsel2  = upsel&&uprstr&& upaddr[1]&&!upaddr[0];
  assign upsel3  = upsel&&uprstr&& upaddr[1]&& upaddr[0];
  assign lupdata  = upsel0? {1'b0,tps_data[30:24]} : 8'bz,
          lupdata  = upsel1?    tps_data[23:16] : 8'bz,
          lupdata  = upsel2?    tps_data[15:8] : 8'bz,
```

```
            lupdata = upsel3?  tps_data[7:0]  : 8'bz;
/*FOLDENDS*/
assign tps_data = tpsreg[52:22];
bch_decode bch1 (.clk(clk), .resync(resync), .in_valid(bch_go[2]),
.in_finalwrite(bch_finalwrite), .in_data(bch_data),
                        .out_valid(bch_valid), .out_data(bch_error));
                        endmodule
```

Listing 22

```
//SccsID = %W% %G%

//FOLDBEGINS 0 0 "Copyright (c) 1997 Pioneer Digital Design Centre Limited ..."
/*--------------------------------------------------------------------
     Copyright (c) 1997 Pioneer Digital Design Centre Limited

NAME:  sydint_rtl.v

PURPOSE: <a one line description>

CREATED: Thu 14 Aug 1997   BY: Paul(Paul McCloy)

MODIFICATION HISTORY:
15/9/97 PK Increased width to 13 to allow for bad_carrier flag

--------------------------------------------------------------------*/
//FOLDENDS

//FOLDBEGINS 0 0 "module symdint ...  <- top level"
//////////////////////////////////////////////////////////////////////
module symdint
//FOLDBEGINS 0 0 "pins ..."
(
    out_data,
    valid,
    d_symbol,

    valid_in,
    demap_data,
    odd_symbol,
    symbol,
    carrier0,
    constellation,

//FOLDBEGINS 0 3 "ram pins ..."
ram_a,
ram_di,
ram_do,
ram_wreq,
//FOLDENDS

//FOLDBEGINS 0 3 "scan pins ..."
tdin,
tdout,
```

```
te,
//FOLDENDS

    nrst,
    clk
);
//FOLDENDS

  parameter WIDTH = 13; // Modified by PK 15/9/97; 12->13
  parameter ADDR_WIDTH = 11;

//FOLDBEGINS 0 2 "outputs ..."
output tdout;

  output valid;
  output [17:0]out_data;
  output d_symbol;

  output [ADDR_WIDTH-1:0]ram_a;
  output [WIDTH-1:0]ram_di;
  output ram_wreq;
  //FOLDENDS
//FOLDBEGINS 0 2 "inputs ..."

  input valid_in;
  input [WIDTH-1:0]demap_data;
  input odd_symbol;
  input symbol;
  input carrier0;
  input [WIDTH-1:0]ram_do;
  input [1:0]constellation;

  input tdin, te;

  input nrst, clk;
  //FOLDENDS
//FOLDBEGINS 0 2 "regs / wires ..."

  //FOLDBEGINS 0 0 "inputs regs ..."

  reg valid_in_reg;
  reg [WIDTH-1:0]demap_data_reg;
  reg odd_symbol_reg;
  reg symbol_reg;
  reg [WIDTH-1:0]ram_do_reg;
  reg [1:0]constellation_reg;
  //FOLDENDS
  //FOLDBEGINS 0 0 "output regs ..."

  reg valid;
  reg [17:0]out_data;
  reg d_symbol;

  reg [ADDR_WIDTH-1:0]ram_a;

  reg [WIDTH-1:0]ram_di;
```

```
reg ram_wreq;
//FOLDENDS

//FOLDBEGINS 0 0 "instate_reg ... "

parameter INSTATE_WAIT_SYMBOL = 2'd0;
parameter INSTATE_WAIT_VALID = 2'd1;
parameter INSTATE_WRITE     = 2'd2;
parameter INSTATE_WRITE_RAM  = 2'd3;

reg [1:0]instate_reg;
//FOLDENDS
//FOLDBEGINS 0 0 "outstate_reg ..."

parameter OUTSTATE_WAIT_WRITEFINISHED   = 3'd0;
parameter OUTSTATE_WAIT0        = 3'd1;
parameter OUTSTATE_WAIT1        = 3'd2;
parameter OUTSTATE_READRAM       = 3'd3;
parameter OUTSTATE_WAIT2        = 3'd4;
parameter OUTSTATE_OUTPUTDATA      = 3'd5;
parameter OUTSTATE_WAIT3        = 3'd6;

reg [2:0]outstate_reg;
//FOLDENDS

reg [ADDR_WIDTH-1:0]read_addr_reg;
reg [WIDTH-1:0]data_reg;
reg next_read_reg, next_write_reg;
reg frist_data_reg;
reg odd_read_reg, odd_write_reg;
reg sym_rst_read_reg, sym_rst_write_reg;

reg [17:0] demapped;
reg [3:0] iminus;
reg [3:0] qminus;
reg [8:0] outi;
reg [8:0] outq;
reg [5:0] demap;


//FOLDBEGINS 0 0 "wires ..."
wire [ADDR_WIDTH-1:0]address_read, address_write;
wire finished_read, finished_write;
wire valid_read, write_valid;

wire [5:0]ini, inq;
//FOLDENDS
//FOLDENDS

ag #(ADDR_WIDTH) r
//FOLDBEGINS 0 2 "pins ..."
(
.address(address_read),

.finished(finished_read),
.next(next_read_reg),
```

```
.random(odd_read_reg),
.sym_rst(sym_rst_read_reg),
.nrst(nrst),
.clk(clk)
   );
   //FOLDENDS

ag #(ADDR_WIDTH) w
//FOLDBEGINS 0 2 "pins ..."
(
.address(address_write),
.finished(finished_write),
.next(next_write_reg),
.random(~odd_write_reg),
.sym_rst(sym_rst_write_reg),
.nrst(nrst),
.clk(clk)
   );
   //FOLDENDS

//FOLDBEGINS 0 2 "latch inputs ..."
always @(posedge clk)
begin
      valid_in_reg   <= valid_in;
      demap_data_reg  <= demap_data;
      odd_symbol_reg  <= odd_symbol;
      symbol_reg    <= symbol;
      ram_do_reg    <= ram_do;
      constellation_reg <= constellation;
   end
   //FOLDENDS

   always @(posedge clk)
   begin
      if( ~nrst )
      //FOLDBEGINS 0 4 "reset ..."
      begin
      instate_reg <= INSTATE_WAIT_SYMBOL;
      outstate_reg <= OUTSTATE_WAIT_WRITEFINISHED;
      next_read_reg <= 0;
      end
      //FOLDENDS
      else
      begin
//FOLDBEGINS 0 4 "input state machine ..."
//$write("DB(%0d %m): instate_reg=%0d    fw=%b\n",
//       $time, instate_reg, finished_write);
case (instate_reg)
         INSTATE_WAIT_SYMBOL: begin
         sym_rst_write_reg <= 1;
         next_write_reg <= 0;
         ram_wreq <= 0;

         if( symbol_reg )
         begin
```

```
        //$write("DB(%0d %m): GOT = %x (NEW SYMBOL)\n", $time,
            demap_data_reg);
        $write("DB(%0d %m): START WRITE\n", $time);
              odd_write_reg <= odd_symbol_reg;
              data_reg <= demap_data_reg;
              instate_reg <= INSTATE_WRITE;
            end
            end
        INSTATE_WAIT_VALID: begin
        ram_wreq <= 0;
        next_write_reg <= 0;
        if( finished_write )
        begin
              $write("DB(%0d %m): END(1) WRITE\n", $time);
              instate_reg <= INSTATE_WAIT_SYMBOL;
            end
            else
            begin
              if( valid_in_reg )
              begin
                data_reg <= demap_data_reg;
                instate_reg <= INSTATE_WRITE;
              end
              end
          end
          INSTATE_WRITE: begin
            sym_rst_write_reg <= 0;
            next_write_reg <= 1;
            ram_a <= address_write;
            //$write("DB(%0d %m): RWrite[%x] = %x\n", $time, address_write,
            data_reg);
            ram_di <= data_reg;
            ram_wreq <= 1;
            if( finished_write )
            begin
              $write("DB(%0d %m): END(2) WRITE\n", $time);
              instate_reg <= INSTATE_WAIT_SYMBOL;
              ram_wreq <= 0;
            end
            else
              instate_reg <= INSTATE_WAIT_VALID;
              end
      endcase
      //FOLDENDS
//FOLDBEGINS 0 4 "output state machine ..."
//$write("DB(%0d %m): outstate_reg=%0d  nr:%b r:%b\n",
//   $time, outstate_reg, next_read_reg, odd_symbol_reg);
case (outstate_reg)
        OUTSTATE_WAIT_WRITEFINISHED: begin
        sym_rst_read_reg <= 1;
        frist_data_reg <= 1;
        valid <= 0;

        if( finished_write )
        begin
              odd_read_reg <= odd_write_reg;
```

```
                outstate_reg <= OUTSTATE_WAIT0;
                $write("DB(%0d %m): START READ\n", $time);
                //$write("DB(%0d %m): Read (NEW SYMBOL)\n", $time,
                address_read);
             end
             end
      OUTSTATE_WAIT0: begin
      sym_rst_read_reg <= 0;
      outstate_reg <= OUTSTATE_WAIT1;
      end
      OUTSTATE_WAIT1: begin
             outstate_reg <= OUTSTATE_READRAM;
             end
             OUTSTATE_READRAM: begin
             //$write("DB(%0d %m): Read [%x]\n", $time, address_read);
             ram_a <= address_read;
             ram_wreq <= 0;
             next_read_reg <= 1;
             outstate_reg <= OUTSTATE_WAIT2;
      end
      OUTSTATE_WAIT2: begin
            next_read_reg <= 0;
            outstate_reg <= OUTSTATE_OUTPUTDATA;
      end
      OUTSTATE_OUTPUTDATA: begin
             out_data <= {outi[8:6], outq[8:6], outi[5:3],
             outq[5:3], outi[2:0], outq[2:0]};
             valid <= 1;
             d_symbol <= frist_data_reg;
             frist_data_reg <= 0;
             outstate_reg <= OUTSTATE_WAIT3;
      end
      OUTSTATE_WAIT3: begin
             valid <= 0;
             if( finished_read )
             begin
                outstate_reg <= OUTSTATE_WAIT_WRITEFINISHED;
                $write("DB(%0d %m): END READ\n", $time);
             end
             else
                outstate_reg <= OUTSTATE_WAIT0;
                end
    endcase
    //FOLDENDS
    end
 end

 always @(constellation_reg or ini or inq)
 //FOLDBEGINS 0 2 "demapper ..."
 begin
 //FOLDBEGINS 0 2 "coarse demapping"

    iminus = {ini[5:3],1'b0} -2'd3;
    qminus = {inq[5:3],1'b0} -2'd3;
    if(constellation_reg==2'b01)
    begin
```

```
        demap  = { 2'b0,
        iminus[2],
        qminus[2],
        !(iminus[2]^iminus[1]),
        !(qminus[2]^qminus[1])
                        };
                        //$writeb(demap,,);
                        //$display(iminus,,ini[5:3]);
      end
      else if(constellation_reg==2'b10)
      begin
        iminus = {ini[5:3],1'b0} -3'd7;
        qminus = {inq[5:3],1'b0} -3'd7;
        demap = { iminus[3],
                        qminus[3],
                        !(iminus[3]^iminus[2]),
                        !(qminus[3]^qminus[2]),
                        (iminus[2]^iminus[1]),
                        (qminus[2]^qminus[1])
                        };
      end
      else
        demap = 6'b0;

      //FOLDENDS

      if(constellation_reg==2'b01)
      begin
    //FOLDBEGINS 0 4 "16QAM"
    if(!iminus[1]&&iminus[0])
    begin
          outi[8:6] = 3'b0;
          outi[5:3] = demap[3]? 3'b111 : 3'b0;
          outi[2:0] = iminus[2]? ini[2:0] : ~ini[2:0];
        end
        else
        begin
          outi[8:6] = 3'b0;
          outi[5:3] = ~ini[2:0];
          outi[2:0] = 3'b111;
        end
        if(!qminus[1]&&qminus[0])
        begin
          outq[8:6] = 3'b0;
          outq[5:3] = demap[2]? 3'b111 : 3'b0;
          outq[2:0] = qminus[2]? inq[2:0] : ~inq[2:0];
        end
        else
        begin
          outq[8:6] = 3'b0;

          outq[5:3] = ~inq[2:0];
          outq[2:0] = 3'b111;
        end

        //FOLDENDS
```

```
        end
        else if(constellation_reg==2'b10)
        begin
    //FOLDBEGINS 0 4 "64QAM"
    if(!iminus[1])
    begin
            outi[8:6] = demap[5]? 3'b111 : 3'b0;
            outi[5:3] = demap[3]? 3'b111 : 3'b0;
            outi[2:0] = iminus[2]? ~ini[2:0] : ini[2:0];
        end
        else if(!iminus[2])
        begin
            outi[8:6] = demap[5]? 3'b111 : 3'b0;
            outi[5:3] = iminus[3]? ini[2:0] : ~ini[2:0];
            outi[2:0] = demap[1]? 3'b111 : 3'b0;
        end
        else
        begin
            outi[8:6] = ~ini[2:0];
            outi[5:3] = demap[3]? 3'b111 : 3'b0;
            outi[2:0] = demap[1]? 3'b111 : 3'b0;
        end
        if(!qminus[1])
        begin
            outq[8:6] = demap[4]? 3'b111 : 3'b0;
            outq[5:3] = demap[2]? 3'b111 : 3'b0;
            outq[2:0] = qminus[2]? ~inq[2:0] : inq[2:0];
        end
        else if(!qminus[2])
        begin
            outq[8:6] = demap[4]? 3'b111 : 3'b0;
            outq[5:3] = qminus[3]? inq[2:0] : ~inq[2:0];
            outq[2:0] = demap[0]? 3'b111 : 3'b0;
        end
        else
        begin
            outq[8:6] = ~inq[2:0];
            outq[5:3] = demap[2]? 3'b111 : 3'b0;
            outq[2:0] = demap[0]? 3'b111 : 3'b0;
        end
        //FOLDENDS
      end
      else
      begin
     //FOLDBEGINS 0 4 "QPSK"
     outi = {6'b0,~ini[2:0]};
     outq = {6'b0,~inq[2:0]};
     //FOLDENDS
      end

      end
      //FOLDENDS

     assign ini = ram_do_reg[11:6];
     assign inq = ram_do_reg[5:0];
```

```
endmodule
//FOLDENDS

//FOLDBEGINS 0 0 "module ag (address gereration)..."
/////////////////////////////////////////////////////////////////
module ag
//FOLDBEGINS 0 0 "pins ..."
(
     address,
     finished,

     next,
     random,
     sym_rst,

     nrst,
     clk
    );
    //FOLDENDS

    parameter ADDR_WIDTH = 12;

 //FOLDBEGINS 0 2 "outputs ..."
 output [ADDR_WIDTH-1:0] address;
 output finished;
 //FOLDENDS
 //FOLDBEGINS 0 2 "inputs ..."
 input next;
 input random;
 input sym_rst;
 input nrst, clk;
 //FOLDENDS

 //FOLDBEGINS 0 2 "regs ..."
 integer i;

    reg finished;
    reg [9:0] prsr_reg;
    reg [11:0] count_reg;

    wire address_valid;
    //FOLDENDS

    always @(posedge clk)
    begin
       if( ~nrst )
       begin
          count_reg <= 0;

          prsr_reg <= 10'd0;
       end
       else
       begin
          if(sym_rst)
          begin
             finished <= 0;
```

```
            count_reg <= 0;
        end
        else
        if( next | (!address_valid & random) )
        begin
            //$write("DB(%0d %m): Next(r:%d)\n", $time, random);
            if( random )
//FOLDBEGINS 0 8 "do the random stuff ..."
begin
                if( !address_valid )
                begin
            //FOLDBEGINS 0 4 "drive the prsr ..."
            if( count_reg == 11'd0 )
                    prsr_reg <= 10'd0;
                    else
                    if( count_reg == 11'd1 )
                    prsr_reg <= 10'd1;
                    else
                    begin
                    for(i=0;i<9;i=i+1)
                    prsr_reg[i] <= prsr_reg[i+1];
                    prsr_reg[9] <= prsr_reg[0] ^ prsr_reg[3];
                    end


                    //FOLDENDS
                    count_reg <= count_reg + 1;
                    //$write("DB(%0d %m): count=%0d Rand(Retry)\n", $time,
                    count_reg);
                end
                else
                begin
                    if( count_reg == 11'd2047 )
                    begin
                        //$write("DB(%0d %m): *** FINISHED Rand\n", $time);
                        finished <= 1;
                        count_reg <= 0;
                        prsr_reg <= 10'd0;
                    end
                    else
                    begin
            //FOLDBEGINS 0 6 "drive the prsr ..."
            if( count_reg == 11'd0 )
                        prsr_reg <= 10'd0;
                        else
                        if( count_reg == 11'd1 )
                        prsr_reg <= 10'd1;

                        else
                        begin
                        for(i=0;i<9;i=i+1)
                        prsr_reg[i] <= prsr_reg[i+1];
                        prsr_reg[9] <= prsr_reg[0] ^ prsr_reg[3];
                        end
                        //FOLDENDS
                        count_reg <= count_reg + 1;
```

```
                    //$write("DB(%0d %m): count=%0d Rand\n", $time, count_reg);
                    finished <= 0;
                end
                end
            end
            //FOLDENDS
            else
//FOLDBEGINS 0 8 "do the sequential stuff ..."
begin
            if( count_reg != 11'd1511 )
            begin
                //$write("DB(%0d %m): count=%0d Sequ\n", $time, count_reg);
                count_reg <= count_reg +1;
                finished <= 0;
            end
            else
            begin
                //$write("DB(%0d %m): *** FINISHED Sequ\n", $time);
                finished <= 1;
                count_reg <= 0;
            end
            end
            //FOLDENDS
        end
        end
  end

//FOLDBEGINS 0 2 "assign address ..."
assign address = (random) ? ({count_reg[0],  // 10
                               prsr_reg[2],  // 9
                               prsr_reg[5],  // 8
                               prsr_reg[8],  // 7
                               prsr_reg[3],  // 6
                               prsr_reg[7],  // 5
                               prsr_reg[0],  // 4
                               prsr_reg[1],  // 3
                               prsr_reg[4],  // 2
                               prsr_reg[6],  // 1
                               prsr_reg[9]}):  // 0
                              count_reg;
                              //FOLDENDS

  assign address_valid = (address < 11'd1512);
  endmodule
  //FOLDENDS
```

Listing 23

```
//SccsID: "@(#)bitdeint.v    1.4 9/14/97"
//FOLDBEGINS 0 0 "Copyright (c) 1997 Pioneer Digital Design Centre Limited"
/***********************************************************
     Copyright (c) 1997 Pioneer Digital Design Centre Limited

  NAME:  bitdeint_rtl.v

  PURPOSE: bit deinterleaver
```

```
CREATED: Wed 23 Jul 1997   BY: Paul(Paul McCloy)

MODIFICATION HISTORY:

*************************************************************/
//FOLDENDS

module bitdeint
//FOLDBEGINS 0 2 "pins ..."
  (
  i_data,
  q_data,
  discard_i,
  discard_q,

    valid,  // output

  //FOLDBEGINS 0 2 "ram0 pins ..."

    ram0_a,
    ram0_di,
    ram0_do,
    ram0_wreq,
    ram0_ce,
    //FOLDENDS
  //FOLDBEGINS 0 2 "ram1 pins ..."

    ram1_a,
    ram1_di,
    ram1_do,
    ram1_wreq,
    ram1_ce,
    //FOLDENDS
  //FOLDBEGINS 0 2 "ram2 pins ..."

    ram2_a,
    ram2_di,
    ram2_do,
    ram2_wreq,
    ram2_ce,
    //FOLDENDS

    bad_carrier,
    valid_in,
    data_in,
    symbol,
    constellation, // constellation
    alpha,  // does not do anything yet

  //FOLDBEGINS 0 2 "scan pins ..."
  tdin,
  tdout,
  te,
  //FOLDENDS

    nrst,
```

```
    clk
  );
  //FOLDENDS

  parameter SBW = 3; // soft bit width

//FOLDBEGINS 0 2 "outputs ..."
//FOLDBEGINS 0 0 "ram0 outputs ..."
output [6:0]ram0_a;
output [((SBW+1)<<1)-1:0]ram0_di;
output ram0_ce;
output ram0_wreq;
//FOLDENDS
//FOLDBEGINS 0 0 "ram1 outputs ..."
output [6:0]ram1_a;
output [((SBW+1)<<1)-1:0]ram1_di;
output ram1_ce;
output ram1_wreq;
//FOLDENDS
//FOLDBEGINS 0 0 "ram2 outputs ..."
output [6:0]ram2_a;
output [((SBW+1)<<1)-1:0]ram2_di;
output ram2_ce;
output ram2_wreq;
//FOLDENDS

   output tdout;

   output [SBW-1:0]i_data;
   output [SBW-1:0]q_data;
   output discard_i;
   output discard_q;

   output valid;

   //FOLDENDS
   //FOLDBEGINS 0 2 "inputs ..."

   input [((SBW+1)<<1)-1:0]ram0_do;
   input [((SBW+1)<<1)-1:0]ram1_do;
   input [((SBW+1)<<1)-1:0]ram2_do;

   input bad_carrier;
   input valid_in;
   input [((SBW<<2)+(SBW<<1))-1:0]data_in;  // 6*SBW bits
   input symbol;
   input [1:0] constellation;
   input [2:0] alpha;

   input tdin, te;

   input nrst, clk;
   //FOLDENDS

 //FOLDBEGINS 0 2 "reg / wire ..."
 //FOLDBEGINS 0 0 "outputs ..."
```

```
//FOLDBEGINS 0 0 "ram0 regs ..."
reg [6:0]ram0_a;
reg [((SBW+1)<<1)-1:0]ram0_di;
reg ram0_ce;
reg ram0_wreq;
//FOLDENDS
//FOLDBEGINS 0 0 "ram1 regs ..."
reg [6:0]ram1_a;
reg [((SBW+1)<<1)-1:0]ram1_di;
reg ram1_ce;
reg ram1_wreq;
//FOLDENDS
//FOLDBEGINS 0 0 "ram2 regs ..."
reg [6:0]ram2_a;
reg [((SBW+1)<<1)-1:0]ram2_di;
reg ram2_ce;
reg ram2_wreq;
//FOLDENDS

    reg [SBW-1:0]i_data;
    reg [SBW-1:0]q_data;
    reg discard_i;
    reg discard_q;

    reg valid;
    //FOLDENDS
    //FOLDBEGINS 0 0 "inputs ..."

    reg valid_in_reg;
    reg [((SBW<<2)+(SBW<<1))-1:0]data_in_reg;  // 6*SBW bits
    reg symbol_reg, bad_carrier_reg;

    reg [1:0] constellation_reg;
    reg [2:0] alpha_reg;
    reg [((SBW+1)<<1)-1:0]ram0_do_reg;
    reg [((SBW+1)<<1)-1:0]ram1_do_reg;
    reg [((SBW+1)<<1)-1:0]ram2_do_reg;

    //FOLDENDS

    reg [6:0]i0_adr_reg;
    reg [6:0]i1_adr_reg;
    reg [6:0]i2_adr_reg;
    reg [6:0]i3_adr_reg;
    reg [6:0]i4_adr_reg;
    reg [6:0]i5_adr_reg;

    reg [2:0] mode_reg;
    reg [(SBW<<2)+(SBW<<1)-1:0]data_reg;  // 6*(SBW) bits
    reg [((SBW+1)<<1)+SBW:0]i_out_buf_reg, q_out_buf_reg; // 3*(SBW+1) bits

    reg ram_filled_reg, out_buf_full_reg, bad_car_reg;

    wire [SBW:0] i0_in, q0_in, i1_in, q1_in ,i2_in ,q2_in;
    wire [SBW:0] i0_ram, q0_ram, i1_ram, q1_ram ,i2_ram ,q2_ram;
    //FOLDENDS
```

```
//FOLDBEGINS 0 2 "latch inputs ..."
always @(posedge clk)
begin
     bad_carrier_reg  <= bad_carrier;
     valid_in_reg     <= valid_in;
     data_in_reg      <= data_in;
     symbol_reg       <= symbol;
     constellation_reg <= constellation;
     alpha_reg        <= alpha;
     ram0_do_reg      <= ram0_do;
     ram1_do_reg      <= ram1_do;
     ram2_do_reg      <= ram2_do;
  end
  //FOLDENDS

  always @(posedge clk)
  begin
    if( ~nrst )
    //FOLDBEGINS 0 4 "reset ..."
    begin
    mode_reg <= 2'b00;
    valid <= 0;
    i0_adr_reg <= 0;
    i1_adr_reg <= 63;
    i2_adr_reg <= 105;
    i3_adr_reg <= 42;
    i4_adr_reg <= 21;
    i5_adr_reg <= 84;


      i_out_buf_reg <= 0;
      q_out_buf_reg <= 0;
      ram_filled_reg <= 0;
      out_buf_full_reg <= 0;
    end
    //FOLDENDS
    else
    begin
      if( valid_in_reg )
      //FOLDBEGINS 0 6 "start cycle ...."
      begin
      data_reg <= data_in_reg;
      bad_car_reg <= bad_carrier_reg;
      //$write("DB(%0d %m): data_reg=%X(%b.%b.%b)\n", $time, data_in_reg,
      //    bad_carrier, bad_carrier_reg, bad_car_reg);
      //FOLDBEGINS 0 2 "logic to read i0,1,2 ..."
      ram0_a <= i0_adr_reg;
      ram0_wreq <= 0;

        ram1_a <= i1_adr_reg;
        ram1_wreq <= 0;

        ram2_a <= i2_adr_reg;
        ram2_wreq <= 0;
        //FOLDENDS
```

```
        ram0_ce <= 1;
        ram1_ce <= (constellation_reg == 2'b10) |
                   (constellation_reg == 2'b01);
                   ram2_ce <= (constellation_reg == 2'b10);

    //FOLDBEGINS 0 2 "output i1 and q1 ..."
    if( out_buf_full_reg & (constellation_reg != 2'b00))
    begin
        valid <= 1;

        i_data <= i_out_buf_reg[((SBW+1)<<1)-2:(SBW+1)];
        discard_i <= i_out_buf_reg[((SBW+1)<<1)-1];

        q_data <= q_out_buf_reg[((SBW+1)<<1)-2:(SBW+1)];
        discard_q <= q_out_buf_reg[((SBW+1)<<1)-1];

        //$write("DB(%0d %m): OUT(1):%x %x\n", $time,
        //      i_out_buf_reg[((SBW+1)<<1)-2:(SBW+1)],
        //      q_out_buf_reg[((SBW+1)<<1)-2:(SBW+1)]);
      end
      //FOLDENDS

      mode_reg <= 3'b001;
      end
      //FOLDENDS
      else
      begin
      //$write("DB(%0d %m): m=%b\n", $time, mode_reg);

      case( mode_reg )
      //FOLDBEGINS 0 8 "3'b001: ... "
      3'b001: begin
      //FOLDBEGINS 0 4 "logic to read q0,1,2 ..."
          ram0_a <= i3_adr_reg;
          ram0_wreq <= 0;

          ram1_a <= i4_adr_reg;
          ram1_wreq <= 0;

          ram2_a <= i5_adr_reg;
          ram2_wreq <= 0;
          //FOLDENDS
          valid <= 0;
          mode_reg <= 3'b010;
          end
      //FOLDENDS
      //FOLDBEGINS 0 8 "3'b010: ..."
      3'b010: begin
      mode_reg <= 3'b011;
      //FOLDBEGINS 0 4 "output i2 and q2 ..."
      if( out_buf_full_reg & (constellation_reg == 2'b10))
      begin
              valid <= 1;

              i_data <= i_out_buf_reg[SBW-1:0];
              discard_i <= i_out_buf_reg[SBW];
```

```
q_data <= q_out_buf_reg[SBW-1:0];
discard_q <= q_out_buf_reg[SBW];

//$write("DB(%0d %m): OUT(2):%x %x\n", $time,
//        i_out_buf_reg[SBW-1:0],
//        q_out_buf_reg[SBW-1:0]);
end
//FOLDENDS
end
//FOLDENDS
//FOLDBEGINS 0 8 "3'b011: ...           "
3'b011: begin
valid <= 0;

//$write("DB(%0d %m): ram read i0:%x i1:%x i2:%x\n",
//        $time,
//        ram0_do_reg[((SBW+1)<<1)-1:SBW+1],
//        ram1_do_reg[((SBW+1)<<1)-1:SBW+1],
//        ram2_do_reg[((SBW+1)<<1)-1:SBW+1]);

i_out_buf_reg <= {ram0_do_reg[((SBW+1)<<1)-1:SBW+1],
ram1_do_reg[((SBW+1)<<1)-1:SBW+1],
ram2_do_reg[((SBW+1)<<1)-1:SBW+1]};

//FOLDBEGINS 0 4 "logic to write new i0,1,2 ..."

ram0_a <= i0_adr_reg;
ram0_wreq <= 1;
ram0_di <= {i0_in, q0_ram};

ram1_a <= i1_adr_reg;
ram1_wreq <= 1;
ram1_di <= {i1_in, q1_ram};

ram2_a <= i2_adr_reg;
ram2_wreq <= 1;
ram2_di <= {i2_in, q2_ram};
//FOLDENDS
mode_reg <= 3'b100;
end
//FOLDENDS
//FOLDBEGINS 0 8 "3'b100: ...           "
3'b100: begin

//$write("DB(%0d %m): ram read q0:%x q1:%x q2:%x\n",
//        $time,
//        ram0_do_reg[SBW:0],
//        ram1_do_reg[SBW:0],
//        ram2_do_reg[SBW:0]);

q_out_buf_reg <= {ram0_do_reg[SBW:0],
ram1_do_reg[SBW:0],
ram2_do_reg[SBW:0]};

out_buf_full_reg <= ram_filled_reg;
//FOLDBEGINS 0 4 "logic to write new q0,1,2 ..."
```

```
            ram0_a <= i3_adr_reg;
            ram0_wreq <= 1;
            ram0_di <= {i0_ram, q0_in};

            ram1_a <= i4_adr_reg;
            ram1_wreq <= 1;
            ram1_di <= {i1_ram, q1_in};

            ram2_a <= i5_adr_reg;
            ram2_wreq <= 1;
            ram2_di <= {i2_ram, q2_in};
            //FOLDENDS

        //FOLDBEGINS 0 4 "output i0 and q0 ..."
        if( out_buf_full_reg )
        begin
                valid <= 1;
                i_data <= i_out_buf_reg[((SBW+1)<<1)+SBW-1:((SBW+1)<<1)];
                discard_i <= i_out_buf_reg[((SBW+1)<<1)+SBW];

                q_data <= q_out_buf_reg[((SBW+1)<<1)+SBW-1:((SBW+1)<<1)];
                discard_q <= q_out_buf_reg[((SBW+1)<<1)+SBW];

                //$write("DB(%0d %m): OUT(0):%x %x\n", $time,

                //      i_out_buf_reg[((SBW+1)<<1)+SBW-1:((SBW+1)<<1)],
                //      q_out_buf_reg[((SBW+1)<<1)+SBW-1:((SBW+1)<<1)]);
            end
            //FOLDENDS
            mode_reg <= 3'b101;
            end
        //FOLDENDS
        //FOLDBEGINS 0 8 "3'b101: ... "
        3'b101:begin
        valid <= 0;
        //FOLDBEGINS 0 4 "increment ram address ..."

            if( i0_adr_reg == 7'd125 )
            begin
              i0_adr_reg <= 0;
              //FOLDBEGINS 0 2 "do i1_adr_reg (63 offset)..."
              i1_adr_reg <= (i1_adr_reg == 7'd20) ? 7'd84 :
              (i1_adr_reg == 7'd41) ? 7'd105 :
              (i1_adr_reg == 7'd62) ? 7'd0 :
              (i1_adr_reg == 7'd83) ? 7'd21 :
              (i1_adr_reg == 7'd104) ? 7'd42 :
                                                        7'd63 ;
                                                        //FOLDENDS
            //FOLDBEGINS 0 2 "do i2_adr_reg (105 offset)..."
            i2_adr_reg <= (i2_adr_reg == 7'd20) ? 7'd42 :
                                 (i2_adr_reg == 7'd41) ? 7'd63 :
                                 (i2_adr_reg == 7'd62) ? 7'd84 :
                                 (i2_adr_reg == 7'd83) ? 7'd105 :
                                 (i2_adr_reg == 7'd104) ? 7'd0 :
                                                        7'd21 ;
                                                        //FOLDENDS
```

```
//FOLDBEGINS 0 2 "do i3_adr_reg (42 offset)..."
i3_adr_reg <= (i3_adr_reg == 7'd20) ? 7'd105 :
                         (i3_adr_reg == 7'd41) ? 7'd0 :
                         (i3_adr_reg == 7'd62) ? 7'd21 :
                         (i3_adr_reg == 7'd83) ? 7'd42 :
                         (i3_adr_reg == 7'd104) ? 7'd63 :
                                                  7'd84 ;
                                                  //FOLDENDS
//FOLDBEGINS 0 2 "do i4_adr_reg (21 offset)..."
i4_adr_reg <= (i4_adr_reg == 7'd20) ? 7'd0 :
                         (i4_adr_reg == 7'd41) ? 7'd21 :
                         (i4_adr_reg == 7'd62) ? 7'd42 :
                         (i4_adr_reg == 7'd83) ? 7'd63 :
                         (i4_adr_reg == 7'd104) ? 7'd84 :
                                                  7'd105 ;
                                                  //FOLDENDS
//FOLDBEGINS 0 2 "do i5_adr_reg (84 offset)..."
i5_adr_reg <= (i5_adr_reg == 7'd20) ? 7'd63 :
                         (i5_adr_reg == 7'd41) ? 7'd84 :
                         (i5_adr_reg == 7'd62) ? 7'd105 :
                         (i5_adr_reg == 7'd83) ? 7'd0 :
                         (i5_adr_reg == 7'd104) ? 7'd21 :

                                                  7'd42 ;
                                                  //FOLDENDS

  ram_filled_reg <= 1;
  end
  else
  begin
  i0_adr_reg <= i0_adr_reg + 1;
  i1_adr_reg <= (i1_adr_reg == 7'd125) ? 0 : i1_adr_reg +1;
  i2_adr_reg <= (i2_adr_reg == 7'd125) ? 0 : i2_adr_reg +1;
  i3_adr_reg <= (i3_adr_reg == 7'd125) ? 0 : i3_adr_reg +1;
  i4_adr_reg <= (i4_adr_reg == 7'd125) ? 0 : i4_adr_reg +1;
  i5_adr_reg <= (i5_adr_reg == 7'd125) ? 0 : i5_adr_reg +1;
 end
 //FOLDENDS
 end
//FOLDENDS
endcase
end
end
end

assign i0_in = { bad_car_reg,
data_reg[(SBW<<2)+(SBW<<1)-1 :(SBW<<2)+SBW]};
assign q0_in = { bad_car_reg,
data_reg[(SBW<<2)+SBW-1   :SBW<<2]};
assign i1_in = { bad_car_reg,
data_reg[(SBW<<2)-1      :(SBW<<1)+SBW]};
assign q1_in = { bad_car_reg,
data_reg[(SBW<<1)+SBW-1   :SBW<<1]};
assign i2_in = { bad_car_reg,
data_reg[(SBW<<1)-1      :SBW]};
assign q2_in = { bad_car_reg,
```

```
  data_reg[SBW-1       :0]};

  assign i0_ram = i_out_buf_reg[((SBW+1)<<1)+SBW:((SBW+1)<<1)];
  assign q0_ram = q_out_buf_reg[((SBW+1)<<1)+SBW:((SBW+1)<<1)];
  assign i1_ram = i_out_buf_reg[((SBW+1)<<1)-1:SBW+1];
  assign q1_ram = q_out_buf_reg[((SBW+1)<<1)-1:SBW+1];
  assign i2_ram = i_out_buf_reg[SBW:0];
  assign q2_ram = q_out_buf_reg[SBW:0];

endmodule
```

Listing 24

```
// SccsId: %W% %G%
/**************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

***************************************************************/


module acc_prod (clk, resync, load, symbol, new_phase, old_phase, xcount,
        acc_out);

 input clk, resync, load, symbol;
 input [10:0] xcount;
 input [13:0] new_phase, old_phase;
 output [29:0] acc_out;

 reg [29:0] acc_out;
 reg [29:0] acc_int;
 reg [14:0] diff;
 reg [25:0] xdiff;

 reg sign;
 reg [14:0] mod_diff;
 reg [25:0] mod_xdiff;



 always @ (posedge clk)
 begin
  if (resync)
  begin
   acc_out <= 0;
   acc_int <= 0;
  end

  else
  begin
   if (load)
    acc_int <= acc_int + {xdiff[25], xdiff[25],    // sign extend
             xdiff[25], xdiff[25], xdiff};
   if (symbol)
   begin
    acc_out <= acc_int;
    acc_int <= 0;
   end
```

```
 end
end

always @ (new_phase or old_phase or xcount)
begin
 diff = {new_phase[13], new_phase}   // sign extend up to allow
    - {old_phase[13], old_phase}; // differences up to 360
 sign = diff[14];
 mod_diff = sign ? (~diff + 1) : diff;
 mod_xdiff = mod_diff * {4'b0, xcount};
 xdiff = sign ? (~mod_xdiff + 1) : mod_xdiff;
end

endmodule
```

Listing 25

```
// SccsId: %W% %G%
/*****************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

*****************************************************************/


module acc_simple (clk, resync, load, symbol, new_phase, old_phase, acc_out);

 input clk, resync, load, symbol;
 input [13:0] new_phase, old_phase;
 output [20:0] acc_out;

 reg [20:0] acc_out;
 reg [20:0] acc_int;
 reg [14:0] diff;


 always @ (posedge clk)
 begin
  if (resync)
  begin
   acc_out <= 0;
   acc_int <= 0;
  end

  else
  begin
   if (load)
    acc_int <= acc_int + {diff[14], diff[14],    // sign extend
            diff[14], diff[14],
            diff[14], diff[14], diff};
   if (symbol)
   begin
    acc_out <= acc_int;
    acc_int <= 0;
   end
  end
```

```
end

always @ (new_phase or old_phase)
 diff = {new_phase[13], new_phase}   // sign extend up to allow
     - {old_phase[13], old_phase}; // differences up to 360

always @ (diff or load)
begin: display

 reg[14:0] real_diff;

 if (load)
 begin
  if (diff[14])
  begin
   real_diff = (~diff + 1);
   $display ("diff = -%0d", real_diff);
  end
  else
   $display ("diff = %0d", diff);
 end
end // display

endmodule
```

Listing 26

```
// SccsId: %W% %G%
/**************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

**************************************************************/


module addr_gen (clk, resync, u_symbol, uc_pilot, got_phase, en, load, guard,
        addr, xcount, guard_reg, symbol);

 input clk, resync, u_symbol, uc_pilot, got_phase;
 input [1:0] guard;
 output en, load, symbol;
 output [1:0] guard_reg;
 output [9:0] addr;
 output [10:0] xcount;

 reg en, load, load_p, inc_count2, symbol;
 reg [1:0] guard_reg;
 reg [5:0] count45;
 reg [10:0] xcount;
 reg [9:0] addr;


 always @ (posedge clk)
 begin
  if (resync)
  begin
   count45 <= 0;
```

```
  load_p <= 0;
  load <= 0;
  inc_count2 <= 0;
  symbol <= 0;
  guard_reg <= 0;
 end

 else
 begin
  if (u_symbol)
  begin
   inc_count2 <= 1;
   guard_reg <= guard;
  end
  if (inc_count2 && uc_pilot)
  begin
   inc_count2 <= 0;
   count45 <= 0;
  end
  if (got_phase)
   count45 <= count45 + 1;
  load_p <= en;
  load <= load_p;
  symbol <= (inc_count2 && uc_pilot);

  addr <= count45;
  en <= got_phase && !resync && (count45 < 45); // !! 45 ?
 end
end

always @ (count45)
 case (count45)
    1: xcount =   1;
    2: xcount =  49;
    3: xcount =  55;
    4: xcount =  88;
    5: xcount = 142;
    6: xcount = 157;
    7: xcount = 193;
    8: xcount = 202;
    9: xcount = 256;
   10: xcount = 280;
   11: xcount = 283;
   12: xcount = 334;
   13: xcount = 433;
   14: xcount = 451;
   15: xcount = 484;
   16: xcount = 526;
   17: xcount = 532;
   18: xcount = 619;
   19: xcount = 637;
   20: xcount = 715;
   21: xcount = 760;
   22: xcount = 766;
   23: xcount = 781;
   24: xcount = 805;
```

```
25: xcount = 874;
26: xcount = 889;
27: xcount = 919;
28: xcount = 940;
29: xcount = 943;
30: xcount = 970;
31: xcount = 985;
32: xcount = 1051;
33: xcount = 1102;
34: xcount = 1108;
35: xcount = 1111;
36: xcount = 1138;
37: xcount = 1141;
38: xcount = 1147;
39: xcount = 1207;
40: xcount = 1270;
41: xcount = 1324;
42: xcount = 1378;
43: xcount = 1492;
44: xcount = 1684;
45: xcount = 1705;
default: xcount = 0;
endcase
endmodule
```

Listing 27

```
// SccsId: %W% %G%
/***************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited
***************************************************************/


module avg_8 (clk, resync, symbol, in_data, avg_out);

 parameter phase_width = 12;

 input clk, resync, symbol;
 input [phase_width-2:0] in_data;
 output [phase_width-2:0] avg_out;

 reg [phase_width-2:0] avg_out;
 reg [phase_width-2:0] store [7:0];


 wire [phase_width-2:0] store7 = store[7];
 wire [phase_width-2:0] store6 = store[6];
 wire [phase_width-2:0] store5 = store[5];
 wire [phase_width-2:0] store4 = store[4];
 wire [phase_width-2:0] store3 = store[3];
 wire [phase_width-2:0] store2 = store[2];
 wire [phase_width-2:0] store1 = store[1];
 wire [phase_width-2:0] store0 = store[0];
```

```
wire [phase_width+1:0] sum = ({store7[phase_width-2], store7[phase_width-2],
store7[phase_width-2], store7}
            + {store6[phase_width-2], store6[phase_width-2], store6[phase_width-2],
store6}
            + {store5[phase_width-2], store5[phase_width-2], store5[phase_width-2],
store5}
            + {store4[phase_width-2], store4[phase_width-2], store4[phase_width-2],
store4}
            + {store3[phase_width-2], store3[phase_width-2], store3[phase_width-2],
store3}
            + {store2[phase_width-2], store2[phase_width-2], store2[phase_width-2],
store2}
            + {store1[phase_width-2], store1[phase_width-2], store1[phase_width-2],
store1}
            + {store0[phase_width-2], store0[phase_width-2], store0[phase_width-2],
store0});

 always @ (posedge clk)
 begin
  if (resync)
  begin
   store[7] <= 0;
   store[6] <= 0;
   store[5] <= 0;
   store[4] <= 0;
   store[3] <= 0;
   store[2] <= 0;
   store[1] <= 0;
   store[0] <= 0;
   avg_out <= 0;
  end
  else if (symbol)
  begin
   store[7] <= store[6];
   store[6] <= store[5];
   store[5] <= store[4];
   store[4] <= store[3];
   store[3] <= store[2];
   store[2] <= store[1];
   store[1] <= store[0];
   store[0] <= in_data;
   avg_out <= sum >> 3;
  end
 end

endmodule
```

Listing 28

```
// SccsId: %W% %G%
/*****************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited
*****************************************************************/


module twowire26 (clk, rst, in_valid, din, out_accept, out_valid, in_accept,
```

```
      dout, set);

input clk, rst, set, in_valid, out_accept;
input [25:0] din;
output in_accept, out_valid;
output [25:0] dout;
reg in_accept, out_valid, acc_int, acc_int_reg, in_valid_reg, val_int;
reg [25:0] dout, din_reg;

always @ (posedge clk)
begin
 if (rst)
  out_valid <= 0;
 else if (acc_int || set)
  out_valid <= val_int;

 if (in_accept)
 begin
  in_valid_reg <= in_valid;
  din_reg <= din;
 end

 if (acc_int)
  dout <= in_accept ? din : din_reg;

 if (set)
  acc_int_reg <= 1;
 else
  acc_int_reg <= acc_int;
end

always @ (out_accept or out_valid or acc_int_reg or in_valid or in_valid_reg)
begin
 acc_int = out_accept || !out_valid;
 in_accept = acc_int_reg || !in_valid_reg;
 val_int = in_accept ? in_valid : in_valid_reg;
end

endmodule


module buffer (clk, nrst, resync, u_symbol_in, uc_pilot_in, ui_data_in,
      uq_data_in, u_symbol_out, uc_pilot_out, ui_data_out,
      uq_data_out, got_phase);

 input clk, nrst, resync, u_symbol_in, uc_pilot_in, got_phase;
 input [11:0] ui_data_in, uq_data_in;
 output u_symbol_out, uc_pilot_out;
 output [11:0] ui_data_out, uq_data_out;

 reg u_symbol_out, uc_pilot_out, accept;
 wire u_symbol_o, uc_pilot_o;
 reg [11:0] ui_data_out, uq_data_out;
 wire [11:0] ui_data_o, uq_data_o;
 wire a, v;
```

```
wire [25:0] d;

wire in_valid = u_symbol_in || uc_pilot_in;
wire rst = !nrst || resync;


twowire26 tw1 (.clk(clk), .rst(rst), .in_valid(in_valid), .din({u_symbol_in,
      uc_pilot_in, ui_data_in, uq_data_in}), .out_accept(a),
      .out_valid(v), .in_accept(), .dout(d), .set(1'b0));

twowire26 tw2 (.clk(clk), .rst(rst), .in_valid(v), .din(d),
      .out_accept(accept), .out_valid(out_valid), .in_accept(a),
      .dout({u_symbol_o, uc_pilot_o, ui_data_o, uq_data_o}),
      .set(1'b0));


always @ (u_symbol_o or uc_pilot_o or ui_data_o or uq_data_o or out_valid or
    accept)
begin
 if (out_valid && accept)
 begin
  u_symbol_out = u_symbol_o;
  uc_pilot_out = uc_pilot_o;
  ui_data_out = ui_data_o;
  uq_data_out = uq_data_o;
 end
 else
 begin
  u_symbol_out = 0;
  uc_pilot_out = 0;
  ui_data_out = 0;
  uq_data_out = 0;
 end
end

always @ (posedge clk)
begin
 if (rst || got_phase)
  accept <= 1;
 else if (uc_pilot_out)
  accept <= 0;
end

endmodule
```

Listing 29

```
// SccsId: %W% %G%
/*****************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

*****************************************************************/


module divide (clk, go, numer, denom, answ, got);
```

```
/*************************************************************************
this divider is optimised on the principal that the answer will always be
less than 1 - ie denom > numer
*************************************************************************/

input clk, go;
input [10:0] numer, denom;
output got;
output [10:0] answ;

reg got;
reg [10:0] answ;
reg [20:0] sub, internal;
reg [3:0] dcount;


always @ (posedge clk)
begin
 if (go)
 begin
  dcount <= 0;
  internal <= numer << 10;
  sub <= denom << 9;
 end
 if (dcount < 11)
 begin
  if (internal > sub)
  begin
   internal <= internal - sub;
   answ[10 - dcount] <= 1;
  end
  else
  begin
   internal <= internal;
   answ[10 - dcount] <= 0;
  end

  sub <= sub >> 1;
  dcount <= dcount + 1;
 end

 got <= (dcount == 10);
end

endmodule
```

Listing 30

```
// SccsId: %W% %G%
/**************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

**************************************************************/


module fserr_str (clk, nrst, resync, u_symbol, uc_pilot, ui_data, uq_data, guard,
```

```
        freq_sweep, sr_sweep, lupdata, upaddr, upwstr, uprstr, upsel1,
        upsel2, ram_di, te, tdin, freq_err, samp_err, ram_rnw,
        ram_addr, ram_do, tdout);

input clk, nrst, resync, u_symbol, uc_pilot, upwstr, uprstr, te, tdin, upsel1,
  upsel2;
input [1:0] guard;
input [3:0] freq_sweep, sr_sweep, upaddr;
input [11:0] ui_data, uq_data;
input [13:0] ram_do;
output ram_rnw, tdout;
output [9:0] ram_addr;
output [12:0] freq_err, samp_err;
output [13:0] ram_di;
inout [7:0] lupdata;

wire got_phase, en, load, symbol, u_symbol_buf, uc_pilot_buf;
wire freq_open, sample_open;
wire [1:0] guard_reg;
wire [10:0] xcount;
wire [11:0] ui_data_buf, uq_data_buf;
wire [13:0] phase_in, phase_out;
wire [20:0] acc_out_simple;
wire [29:0] acc_out_prod;
wire [12:0] freq_err_uf, samp_err_uf;
wire [12:0] freq_err_fil, samp_err_fil, freq_twiddle,
        sample_twiddle;


buffer buffer (.clk(clk), .nrst(nrst), .resync(resync), .u_symbol_in(u_symbol),
      .uc_pilot_in(uc_pilot), .ui_data_in(ui_data),
      .uq_data_in(uq_data), .u_symbol_out(u_symbol_buf),
      .uc_pilot_out(uc_pilot_buf), .ui_data_out(ui_data_buf),
      .uq_data_out(uq_data_buf), .got_phase(got_phase));

tan_taylor phase_extr (.clk(clk), .nrst(nrst), .resync(resync),
        .uc_pilot(uc_pilot_buf), .ui_data(ui_data_buf),
        .uq_data(uq_data_buf), .phase(phase_in),
        .got_phase(got_phase));

addr_gen addr_gen (.clk(clk), .resync(resync), .u_symbol(u_symbol_buf),
       .uc_pilot(uc_pilot_buf), .got_phase(got_phase), .en(en),
       .load(load), .guard(guard), .addr(ram_addr), .xcount(xcount),
       .guard_reg(guard_reg), .symbol(symbol));

pilot_store pilot_store (.clk(clk), .en(en), .ram_do(ram_do),
         .phase_in(phase_in), .ram_rnw(ram_rnw),
         .ram_di(ram_di), .phase_out(phase_out));

acc_simple acc_simple (.clk(clk), .resync(resync), .load(load),
        .symbol(symbol), .new_phase(phase_in),
        .old_phase(phase_out), .acc_out(acc_out_simple));

acc_prod acc_prod (.clk(clk), .resync(resync), .load(load),
       .symbol(symbol), .new_phase(phase_in),
       .old_phase(phase_out), .xcount(xcount),
```

```
        .acc_out(acc_out_prod));

slow_arith slow_arith (.acc_simple(acc_out_simple), .acc_prod(acc_out_prod),
        .guard(guard_reg), .freq_err_uf(freq_err_uf),
        .samp_err_uf(samp_err_uf));

avg_8 #(14)
    lpf_freq (.clk(clk), .resync(resync), .symbol(symbol),
        .in_data(freq_err_uf), .avg_out(freq_err_fil));

avg_8 #(14)
    lpf_samp (.clk(clk), .resync(resync), .symbol(symbol),
        .in_data(samp_err_uf), .avg_out(samp_err_fil));

/* median_filter #(14)
    lpf_freq (.clk(clk), .nrst(nrst), .in_valid(symbol),
        .din(freq_err_uf), .dout(freq_err_fil));

median_filter #(14)
    lpf_samp (.clk(clk), .nrst(nrst), .in_valid(symbol),
        .din(samp_err_uf), .dout(samp_err_fil));    */


sweep_twiddle sweep_twiddle (.freq_err_fil(freq_err_fil),
            .samp_err_fil(samp_err_fil),
            .freq_sweep(freq_sweep),
            .sr_sweep(sr_sweep), .freq_open(freq_open),
            .sample_open(sample_open),
            .freq_twiddle(freq_twiddle),
            .sample_twiddle(sample_twiddle),
            .freq_err_out(freq_err),
            .samp_err_out(samp_err));

lupidec lupidec (.clk(clk), .nrst(nrst), .resync(resync), .upaddr(upaddr),
        .upwstr(upwstr), .uprstr(uprstr), .lupdata(lupdata),
        .freq_open(freq_open), .sample_open(sample_open),
        .freq_twiddle(freq_twiddle), .sample_twiddle(sample_twiddle),
        .sample_loop_bw(), .freq_loop_bw(), .freq_err(freq_err),
        .samp_err(samp_err), .f_err_update(), .s_err_update());

endmodule
```

Listing 31

```
// SccsId: %W% %G%
/**************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited
**************************************************************/


module lupidec (clk, nrst, resync, upaddr, upwstr, uprstr, lupdata, freq_open,
    sample_open, freq_twiddle, sample_twiddle, sample_loop_bw,
    freq_loop_bw, freq_err, samp_err, f_err_update,
    s_err_update);

 input clk, nrst, resync, upwstr, uprstr, f_err_update, s_err_update;
```

```
input [3:0] upaddr;
input [12:0] freq_err, samp_err;
inout [7:0] lupdata;
output freq_open, sample_open;
output [12:0] freq_twiddle, sample_twiddle, sample_loop_bw, freq_loop_bw;

reg freq_open, sample_open;
reg [12:0] freq_twiddle, sample_twiddle, sample_loop_bw, freq_loop_bw;

wire wr_str;
wire [3:0] wr_addr;
wire [7:0] wr_data;


/*FOLDBEGINS 0 2 "address decode"*/
 /*FOLDBEGINS 0 0 "read decode"*/
 wire f_err_h_ren = (upaddr == 4'he);
 wire f_err_l_ren = (upaddr == 4'hf);
 wire s_err_h_ren = (upaddr == 4'hc);
 wire s_err_l_ren = (upaddr == 4'hd);
 wire f_twd_h_ren = (upaddr == 4'h4);
 wire f_twd_l_ren = (upaddr == 4'h5);
 wire s_twd_h_ren = (upaddr == 4'h8);
 wire s_twd_l_ren = (upaddr == 4'h9);
 wire f_lbw_h_ren = (upaddr == 4'h6);
 wire f_lbw_l_ren = (upaddr == 4'h7);
 wire s_lbw_h_ren = (upaddr == 4'ha);
 wire s_lbw_l_ren = (upaddr == 4'hb);
 /*FOLDENDS*/

 /*FOLDBEGINS 0 0 "write decode"*/
 wire f_twd_h_wen = (wr_addr == 4'h4);
 wire f_twd_l_wen = (wr_addr == 4'h5);
 wire s_twd_h_wen = (wr_addr == 4'h8);
 wire s_twd_l_wen = (wr_addr == 4'h9);
 wire f_lbw_h_wen = (wr_addr == 4'h6);
 wire f_lbw_l_wen = (wr_addr == 4'h7);
 wire s_lbw_h_wen = (wr_addr == 4'ha);
 wire s_lbw_l_wen = (wr_addr == 4'hb);
 /*FOLDENDS*/
 /*FOLDENDS*/

 /*FOLDBEGINS 0 2 "upi regs"*/
  /*FOLDBEGINS 0 0 "freq error status reg "*/
  upi_status_reg2 fr_err (.clk(clk), .nrst(nrst), .status_value({3'b0, freq_err}),
           .capture_strobe(f_err_update), .read_strobe(uprstr),
           .reg_select_l(f_err_l_ren), .reg_select_h(f_err_h_ren),
           .lupdata(lupdata));
  /*FOLDENDS*/

  /*FOLDBEGINS 0 0 "sample error status reg"*/
  upi_status_reg2 sr_err (.clk(clk), .nrst(nrst), .status_value({3'b0, samp_err}),
           .capture_strobe(s_err_update), .read_strobe(uprstr),
           .reg_select_l(s_err_l_ren), .reg_select_h(s_err_h_ren),
           .lupdata(lupdata));
  /*FOLDENDS*/
```

```
/*FOLDBEGINS 0 0 "control regs write latch"*/
upi_write_latch #(3)
    write_lat (.clk(clk), .nrst(nrst), .lupdata(lupdata), .upaddr(upaddr),
        .write_strobe(upwstr), .write_data(wr_data),
        .write_address(wr_addr), .write_sync(wr_str));
/*FOLDENDS*/

/*FOLDBEGINS 0 0 "freq twiddle etc rdbk regs"*/
upi_rdbk_reg freq_r_upper (.control_value({freq_open, 2'b0, freq_twiddle[12:8]}),
        .read_strobe(uprstr), .reg_select(f_twd_h_ren),
        .lupdata(lupdata));

upi_rdbk_reg freq_r_lower (.control_value(freq_twiddle[7:0]), .read_strobe(uprstr),
        .reg_select(f_twd_l_ren), .lupdata(lupdata));
/*FOLDENDS*/

/*FOLDBEGINS 0 0 "samp twiddle etc rdbk regs"*/
upi_rdbk_reg samp_r_upper (.control_value({sample_open, 2'b0,
sample_twiddle[12:8]}),
        .read_strobe(uprstr), .reg_select(s_twd_h_ren),
        .lupdata(lupdata));

upi_rdbk_reg samp_r_lower (.control_value(sample_twiddle[7:0]),
.read_strobe(uprstr),
        .reg_select(s_twd_l_ren), .lupdata(lupdata));
/*FOLDENDS*/

/*FOLDBEGINS 0 0 "freq loop bw rdbk regs"*/
upi_rdbk_reg fr_lp_r_upper (.control_value({3'b0, freq_loop_bw[12:8]}),
        .read_strobe(uprstr), .reg_select(f_lbw_h_ren),
        .lupdata(lupdata));

upi_rdbk_reg fr_lp_r_lower (.control_value(freq_loop_bw[7:0]),
        .read_strobe(uprstr), .reg_select(f_lbw_l_ren),
        .lupdata(lupdata));
/*FOLDENDS*/

/*FOLDBEGINS 0 0 "samp loop bw rdbk regs"*/
upi_rdbk_reg sr_lp_r_upper (.control_value({3'b0, sample_loop_bw[12:8]}),
        .read_strobe(uprstr), .reg_select(s_lbw_h_ren),
        .lupdata(lupdata));

upi_rdbk_reg sr_lp_r_lower (.control_value(sample_loop_bw[7:0]),
        .read_strobe(uprstr), .reg_select(s_lbw_l_ren),
        .lupdata(lupdata));
/*FOLDENDS*/
/*FOLDENDS*/

/*FOLDBEGINS 0 2 "control regs"*/
 always @ (posedge clk)
 begin
  if (!nrst)
  begin
   freq_open <= 0;
   sample_open <= 0;
```

```
 freq_twiddle <= 0;
 sample_twiddle <= 0;
 sample_loop_bw <= 0;  //????
 freq_loop_bw <= 0;   //????
end
else
begin
 if (wr_str)
 begin
  if (f_twd_h_wen)
  begin
   freq_open <= wr_data[7];
   freq_twiddle[12:8] <= wr_data[4:0];
  end

  if (f_twd_l_wen)
   freq_twiddle[7:0] <= wr_data[7:0];

  if (s_twd_h_wen)
  begin
   sample_open <= wr_data[7];
   sample_twiddle[12:8] <= wr_data[4:0];
  end

  if (s_twd_l_wen)
   sample_twiddle[7:0] <= wr_data[7:0];

  if (f_lbw_h_wen)
   freq_loop_bw[12:8] <= wr_data[4:0];

  if (f_lbw_l_wen)
   freq_loop_bw[7:0] <= wr_data[7:0];

  if (s_lbw_h_wen)
   sample_loop_bw[12:8] <= wr_data[4:0];

  if (s_lbw_l_wen)
   sample_loop_bw[7:0] <= wr_data[7:0];

 end
end
end
/*FOLDENDS*/

endmodule
```

Listing 32

```
// SccsId: %W% %G%
/*******************************************************************
 Copyright (c) 1997 Pioneer Digital Design Centre Limited
********************************************************************/
```

```
module pilot_store (clk, en, ram_do, phase_in, ram_rnw, ram_di, phase_out);

 input clk, en;
 // input [9:0] addr;
 input [13:0] phase_in;
 input [13:0] ram_do;
 output ram_rnw;
 output [13:0] ram_di, phase_out;

 wire ram_rnw;
 // reg en_d1;
 // reg [9:0] addr_reg;
 // reg [13:0] mem [579:0];
 reg [13:0] phase_out;  //, phase_in_reg;
 wire [13:0] ram_di;


 always @ (posedge clk)
 begin
  // en_d1 <= en;

  if (en)
  begin
 //  phase_in_reg <= phase_in;
 //  addr_reg <= addr;
   phase_out <= ram_do;
 //  phase_out <= mem[addr];
  end
 // if (en_d1)
 //  mem[addr_reg] <= phase_in_reg;
 end

 assign ram_di = phase_in;
 assign ram_rnw = !en;

 endmodule
```

Listing 33

```
// SccsId: %W% %G%
/***************************************************************
  Copyright (c) 1997 Pioneer Digital Design Centre Limited

****************************************************************/


module slow_arith (acc_simple, acc_prod, guard, freq_err_uf, samp_err_uf);

  input [1:0] guard;
  input [20:0] acc_simple;
  input [29:0] acc_prod;
  output [12:0] freq_err_uf, samp_err_uf;

  reg [12:0] freq_err_uf, samp_err_uf;
  reg [20:0] freq_scale;
  reg [38:0] inter_freq;
```

```
reg sign;
reg [20:0] mod_acc;
reg [38:0] mod_trunc_sat;
reg [41:0] mod;

reg sign_a, sign_b, sign_inter_sr;
reg [20:0] mod_acc_s;
reg [29:0] mod_acc_p;
reg [35:0] a, mod_a;
reg [35:0] b, mod_b;
reg [36:0] mod_diff, diff;
reg [46:0] inter_sr, mod_inter_sr;


parameter sp = 45, acc_x = 33927, samp_scale = 11'b10100100110;

always @ (guard)
 case (guard)
  2'b00: freq_scale = 21'b011110100111110001011; // guard == 64
  2'b01: freq_scale = 21'b011101101110001000011; // guard == 128
  2'b10: freq_scale = 21'b011100000100011101010; // guard == 256
  2'b11: freq_scale = 21'b011001010000110011111; // guard == 512
 endcase

always @ (acc_simple or freq_scale)
begin

 sign = acc_simple[20];
 mod_acc = sign ? (~acc_simple + 1) : acc_simple;
 mod = (freq_scale * mod_acc);
// inter_freq = sign ? (~mod + 1) : mod;

 if (mod[41:38] > 0)
 begin
  mod_trunc_sat = 39'h3fffffffff;
  $display("freq_err saturated");
 end
 else
  mod_trunc_sat = mod[38:0];

 inter_freq = sign ? (~mod_trunc_sat + 1) : mod_trunc_sat;

 freq_err_uf = inter_freq >> 26;
end

always @ (acc_simple or acc_prod)
begin

 sign_a = acc_prod[29];
 mod_acc_p = sign_a ? (~acc_prod + 1) : acc_prod;
 mod_a = sp * mod_acc_p;
 a = sign_a ? (~mod_a + 1) : mod_a;

 sign_b = acc_simple[20];
 mod_acc_s = sign_b ? (~acc_simple + 1) : acc_simple;
 mod_b = acc_x * mod_acc_s;
```

```
 b = sign_b ? (~mod_b + 1 ) : mod_b;

 diff = {a[35], a} - {b[35], b};   // sign extend

 sign_inter_sr = diff[36];
 mod_diff = sign_inter_sr ? (~diff + 1) : diff;
 mod_inter_sr = (mod_diff * samp_scale);
 inter_sr = sign_inter_sr ? (~mod_inter_sr + 1) : mod_inter_sr;

 samp_err_uf = inter_sr >> 34; //!!scaling!!
end

endmodule
```

Listing 34

```
// SccsId: %W% %G%
/****************************************************************
 Copyright (c) 1997 Pioneer Digital Design Centre Limited

***************************************************************/
module sweep_twiddle (freq_err_fil, samp_err_fil, freq_sweep, sr_sweep,
        freq_open, sample_open, freq_twiddle, sample_twiddle,
        freq_err_out, samp_err_out);

 input freq_open, sample_open;
 input [3:0] freq_sweep, sr_sweep;
 input [12:0] freq_err_fil, samp_err_fil, freq_twiddle, sample_twiddle;
 output [12:0] freq_err_out, samp_err_out;

 reg [12:0] freq_err_out, samp_err_out;
 reg [12:0] freq_err_swept, samp_err_swept;


 always @ (freq_sweep or freq_err_fil)
 case (freq_sweep)
  4'b0000: freq_err_swept = freq_err_fil;
  4'b0001: freq_err_swept = freq_err_fil + 500;
  4'b0010: freq_err_swept = freq_err_fil + 1000;
  4'b0011: freq_err_swept = freq_err_fil + 1500;
  4'b0100: freq_err_swept = freq_err_fil + 2000;
  4'b0101: freq_err_swept = freq_err_fil + 2500;
  4'b0110: freq_err_swept = freq_err_fil + 3000;
  4'b0111: freq_err_swept = freq_err_fil + 3500;
  default: freq_err_swept = freq_err_fil;
 endcase

 always @ (sr_sweep or samp_err_fil)
 case (sr_sweep)
  4'b0000: samp_err_swept = samp_err_fil;
  4'b0001: samp_err_swept = samp_err_fil + 500;
  4'b0010: samp_err_swept = samp_err_fil - 500;
  4'b0011: samp_err_swept = samp_err_fil + 1000;
  4'b0100: samp_err_swept = samp_err_fil - 1000;
  4'b0101: samp_err_swept = samp_err_fil + 1500;
  4'b0110: samp_err_swept = samp_err_fil - 1500;
  4'b0111: samp_err_swept = samp_err_fil + 2000;
```

```
 4'b1000: samp_err_swept = samp_err_fil - 2000;
 default: samp_err_swept = samp_err_fil;
endcase

always @ (freq_err_swept or freq_open or freq_twiddle)
if (freq_open)
 freq_err_out = freq_twiddle;
else
 freq_err_out = freq_err_swept + freq_twiddle;

always @ (samp_err_swept or sample_open or sample_twiddle)
if (sample_open)
 samp_err_out = sample_twiddle;
else
 samp_err_out = samp_err_swept + sample_twiddle;


endmodule
```

Listing 35

```
// SccsId: %W% %G%
/*****************************************************************
 Copyright (c) 1997 Pioneer Digital Design Centre Limited

******************************************************************/



module tan_taylor (clk, nrst, resync, uc_pilot, ui_data, uq_data, phase,
        got_phase);

 input clk, nrst, resync, uc_pilot;
 input [11:0] ui_data, uq_data;
 output got_phase;
 output [13:0] phase;

 reg got_phase;
 reg [13:0] phase;
 reg add, qgti, modqeqi, i_zero_reg, q_zero_reg, go;
 reg [1:0] quadrant;
 reg [6:0] count, count_d1;
 reg [10:0] mod_i, mod_q, coeff, numer, denom;
 reg [21:0] x_sqd, x_pow, next_term, sum, flip, next_term_unshift, prev_sum,
      x_sqd_unshift, x_pow_unshift;
 wire got;
 wire [10:0] div;

 parameter pi = 6434, pi_over2 = 3217, minus_pi_o2 = 13167, pi_over4 = 1609;


 divide div1 (clk, go, numer, denom, div, got);

 always @ (posedge clk)
 begin
  if (!nrst || resync)
   count <= 7'b1111111;
```

```
else
begin
 if (uc_pilot)
 begin
  mod_i <= ui_data[11] ? (~ui_data[10:0] + 1) : ui_data[10:0];
  mod_q <= uq_data[11] ? (~uq_data[10:0] + 1) : uq_data[10:0];
  quadrant <= {uq_data[11], ui_data[11]};
  count <= 0;
  go <= 0;
 end

 else
 begin
  if (count == 0)
  begin
   qgti <= (mod_q > mod_i);
   modqeqi <= (mod_q == mod_i);
   i_zero_reg <= (mod_i == 0);
   q_zero_reg <= (mod_q == 0);
   add <= 0;
   go <= 1;
   count <= 1;
  end

  if ((count >= 3) && (count < 71))
   count <= count + 2;

  if (count == 1)
  begin
   go <= 0;
   if (got)
   begin
    sum <= div;
    x_pow <= div;
    x_sqd <= x_sqd_unshift >> 11;
    count <= 3;
   end
  end

  if ((count > 1) && (count < 69))
   x_pow <= x_pow_unshift >> 11;
  if ((count > 3) && (count < 69))
   next_term <= next_term_unshift >> 12;
  if ((count > 5) && (count < 69))
  begin
   prev_sum <= sum;
   sum <= add ? (sum + next_term) : (sum - next_term);
   add <= !add;
  end
 end
 if (count == 67)
  sum <= (prev_sum + sum) >> 1;
 if (count == 69)
 casex ({i_zero_reg, q_zero_reg, qgti, modqeqi, quadrant})
  6'b1xx0_0x: phase <= pi_over2;
  6'b1xx0_1x: phase <= minus_pi_o2;
```

```
    6'b01x0_x0: phase <= 0;
    6'b01x0_x1: phase <= pi;

    6'b0010_00: phase <= {2'b00, flip[11:0]};
    6'b0010_01: phase <= pi - {2'b00, flip[11:0]};
    6'b0010_10: phase <= 0 - {2'b00, flip[11:0]};
    6'b0010_11: phase <= {2'b00, flip[11:0]} - pi;

    6'b0000_00: phase <= {2'b00, sum[11:0]};
    6'b0000_01: phase <= pi - {2'b00, sum[11:0]};
    6'b0000_10: phase <= 0 - {2'b00, sum[11:0]};
    6'b0000_11: phase <= {2'b00, sum[11:0]} - pi;

    6'bxxx1_00: phase <= pi_over4;
    6'bxxx1_01: phase <= pi - pi_over4;
    6'bxxx1_10: phase <= 0 - pi_over4;
    6'bxxx1_11: phase <= pi_over4 - pi;
   endcase

   count_d1 <= count;
   got_phase <= (count == 69);
  end
 end

 always @ (div)
  x_sqd_unshift = div * div; // had to do this in order to stop synthesis throwing away!

 always @ (x_pow or coeff)
  next_term_unshift = (x_pow * coeff); // compass dp_cell mult_booth_csum

 always @ (x_pow or x_sqd)
  x_pow_unshift = (x_pow * x_sqd);   // compass dp_cell mult_booth_csum

 always @ (count_d1)
 case (count_d1)
    3: coeff = 11'b10101010101;
    5: coeff = 11'b01100110011;
    7: coeff = 11'b01001001001;
    9: coeff = 11'b00111000111;
    11: coeff = 11'b00101110100;
    13: coeff = 11'b00100111011;
    15: coeff = 11'b00100010001;
    17: coeff = 11'b00011110001;
    19: coeff = 11'b00011010111;
    21: coeff = 11'b00011000011;
    23: coeff = 11'b00010110010;
    25: coeff = 11'b00010100011;
    27: coeff = 11'b00010010111;
    29: coeff = 11'b00010001101;
    31: coeff = 11'b00010000100;
    33: coeff = 11'b00001111100;
    35: coeff = 11'b00001110101;
    37: coeff = 11'b00001101110;
    39: coeff = 11'b00001101001;
    41: coeff = 11'b00001100100;
    43: coeff = 11'b00001011111;
```

```
    45: coeff = 11'b00001011011;
    47: coeff = 11'b00001010111;
    49: coeff = 11'b00001010011;
    51: coeff = 11'b00001010000;
    53: coeff = 11'b00001001101;
    55: coeff = 11'b00001001010;
    57: coeff = 11'b00001000111;
    59: coeff = 11'b00001000101;
    61: coeff = 11'b00001000011;
    63: coeff = 11'b00001000001;
   // 65: coeff = 11'b00000111111;
   // 67: coeff = 11'b00000111101;
   // 69: coeff = 11'b00000111011;
   // 71: coeff = 11'b00000111001;
   // 73: coeff = 11'b00000111000;
   // 75: coeff = 11'b00000110110;
   // 77: coeff = 11'b00000110101;
  default: coeff = 11'bx;
 endcase

 always @ (mod_q or mod_i or qgti)
 begin
  numer = qgti ? mod_i : mod_q;
  denom = qgti ? mod_q : mod_i;
 end

 always @ (sum)
  flip = pi_over2 - sum;

// always @ (got)
// if (got)
//  $display("numer was %d, denom was %d, div then %d", numer, denom, div);

// always @ (count)
// if (count < 68 ) $display("as far as x to the %0d term, approx = %d", (count-6),
sum);

 always @ (got_phase)
  begin: display
  reg [13:0] real_phase;

   if (phase[13])
   begin
    real_phase = (~phase + 1);
    if (got_phase) $display("%t: got phase, phase = -%0d", $time, real_phase);
   end
   else
   begin
    if (got_phase) $display("%t: got phase, phase = %0d", $time, phase);
   end
  end // display

 endmodule
```

While this invention has been explained with reference to the structure disclosed herein, it is not confined to the details set forth and this application is intended to cover any modifications and changes as may come within the scope of the following claims:

CLAIMS

1. A digital receiver for multicarrier signals comprising:

an amplifier accepting an analog multicarrier signal, wherein said multicarrier signal comprises a stream of data symbols having a symbol period T_s, wherein the symbols comprise an active interval, a guard interval, and a boundary therebetween, said guard interval being a replication of a portion of said active interval;

an analog to digital converter coupled to said amplifier;

an I/Q demodulator for recovering in phase and quadrature components from data sampled by said analog to digital converter;

an automatic gain control circuit coupled to said analog to digital converter for providing a gain control signal for said amplifier;

a low pass filter circuit accepting I and Q data from said I/Q demodulator, wherein said I and Q data are decimated;

a resampling circuit receiving said decimated I and Q data at a first rate and outputting resampled I and Q data at a second rate;

an FFT window synchronization circuit coupled to said resampling circuit for locating a boundary of said guard interval;

a real-time pipelined FFT processor operationally associated with said FFT window synchronization circuit, wherein said FFT processor comprises at least one stage, said stage comprising:

a complex coefficient multiplier; and

a memory having a lookup table defined therein for multiplicands being multiplied in said complex coefficient multiplier, a value of each said multiplicand being unique in said lookup table; and

a monitor circuit responsive to said FFT window synchronization circuit for detecting a predetermined event, whereby said event indicates that a boundary between an active symbol and a guard interval has been located.

2. The receiver according to claim 1, wherein said FFT window synchronization circuit comprises:

a first delay element accepting currently arriving resampled I and Q data, and outputting delayed resampled I and Q data;

a subtracter, for producing a difference signal representative of a difference between said currently arriving resampled I and Q data and said delayed resampled I and Q data;

a first circuit for producing an output signal having a unipolar magnitude that is representative of said difference signal of said subtracter;

a second delay element for storing said output signal of said first circuit;

a third delay element receiving delayed output of said second delay element; and

a second circuit for calculating a statistical relationship between data stored in said second delay element and data stored in said third delay element and having an output representative of said statistical relationship.

3. The receiver according to claim 2, wherein said statistical relationship comprises an F ratio.

4. The receiver according to claim 1, wherein said FFT processor operates in an 8K mode.

5. The receiver according to claim 1, wherein said wherein said FFT processor further comprises an address generator for said memory, said address generator accepting a signal representing an order dependency of a currently required multiplicand, and outputting an address of said memory wherein said currently required multiplicand is stored.

6. The receiver according to claim 5, wherein each said multiplicand is stored in said lookup table in order of its respective order dependency for multiplication by said complex coefficient multiplier, said order dependencies of said multiplicands defining an incrementation sequence, and said address generator comprises:

an accumulator for storing a previous address that was generated by said address generator;

a circuit for calculating an incrementation value of said currently required multiplicand; and

an adder for adding said incrementation value to said previous address.

7. The receiver according to claim 6, wherein said lookup table comprises a plurality of rows, and said incrementation sequence comprises a plurality of incrementation sequences, said multiplicands being stored in row order, wherein

in a first row a first incrementation sequence is 0;

in a second row a second incrementation sequence is 1;

in a third row first and second break points B1, B2 of a third incrementation sequence are respectively determined by the relationships

$$B1_{M_N} = 4^N B1_{M_N} - \sum_{n=0}^{N-1} 4^n$$

$$B2_{M_N} = \sum_{n=0}^{N} 4^n$$

; and

in a fourth row a third break point B3 of a third incrementation sequence is determined by the relationship

$$B3_{M_N} = 2 \times 4^N + 2$$

wherein M_N represents the memory of an Nth stage of said FFT processor.

8. The receiver according to claim 1, further comprising channel estimation and correction circuitry comprising:

pilot location circuitry receiving a transformed digital signal representing a frame from said FFT processor for locating pilot carriers therein, wherein said pilot carriers are spaced apart in a carrier spectrum of said transformed digital signal at intervals K and have predetermined magnitudes, said pilot location circuitry comprising:

a first circuit for computing an order of carriers in said transformed digital signal modulo K;

K accumulators coupled to said second circuit for accumulating magnitudes of said carriers in said transformed digital signal, said accumulated magnitudes defining a set; and

a correlation circuit for correlating K sets of accumulated magnitude values with said predetermined magnitudes, wherein a first member having a position calculated modulo K in each of said K sets is uniquely offset from a start position of said frame.

9. The receiver according to claim 8, wherein said pilot location circuitry further comprises a bit reversal circuit for reversing a bit order of said transformed digital signal.

10. The receiver according to claim 7, wherein said magnitudes of said carriers and said predetermined magnitudes are amplitudes.

11. The receiver according to claim 7, wherein said magnitudes of said carriers and said predetermined magnitudes are absolute values.

12. The receiver according to claim 7, wherein said correlation circuitry further comprises a peak tracking circuit for determining a spacing between a first peak and a second peak of said K sets of accumulated magnitudes.

13. The receiver according to claim 7, wherein said channel estimation and correction circuitry further comprises:

an interpolating filter for estimating a channel response between said pilot carriers; and

a multiplication circuit for multiplying data carriers output by said FFT processor with a correction coefficient produced by said interpolating filter.

14. The receiver according to claim 7, wherein said channel estimation and correction circuitry further comprises

a phase extraction circuit accepting a data stream of phase-uncorrected I and Q data from said FFT processor, and producing a signal representative of a phase angle of said uncorrected data, said phase extraction circuit including an accumulator for accumulating the phase angles of succeeding phase-uncorrected I and Q data.

15. The receiver according to claim 14, said channel estimation and correction circuitry further comprises:

an automatic frequency control circuit coupled to said phase extraction circuit and said accumulator, comprising;

a memory for storing an accumulated common phase error of a first symbol carried in said phase-uncorrected I and Q data;

wherein said accumulator is coupled to said memory and accumulates a difference between a common phase error of a plurality of pilot carriers in a second symbol and a common phase error of corresponding pilot carriers in said first symbol;

an output of said accumulator being coupled to said I/Q demodulator.

16. The receiver according to claim 15, wherein said coupled output of said accumulator is enabled in said I/Q demodulator only during reception of a guard interval therein.

17. The receiver according to claim 14, said channel estimation and correction circuitry further comprises an automatic sampling rate control circuit coupled to said phase extraction circuit, comprising:

a memory for storing accumulated phase errors of pilot carriers in a first symbol carried in said phase-uncorrected I and Q data;

wherein said accumulator is coupled to said memory and accumulates differences between phase errors of pilot carriers in a second symbol and phase errors of corresponding pilot carriers in said first symbol to define a plurality of accumulated intersymbol carrier phase error differentials, a phase slope being defined by a difference between a first accumulated intersymbol carrier phase differential and a second accumulated intersymbol carrier phase differential;

an output of said accumulator being coupled to said I/Q demodulator.

18. The receiver according to claim 17, wherein said sampling rate control circuit stores a plurality of accumulated intersymbol carrier phase error differentials and computes a line of best fit therebetween.

19. The receiver according to claim 17, wherein said coupled output signal of said accumulator is enabled in said resampling circuit only during reception of a guard interval therein.

20. The receiver according to claim 17, wherein a common memory for storing output of said phase extraction circuit is coupled to said automatic frequency control circuit and to said automatic sampling rate control circuit.

21. The receiver according to claim 14, wherein said phase extraction circuit further comprises:

a pipelined circuit for iteratively computing the arctangent of an angle of rotation according to the series

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots, \qquad |x| < 1$$

wherein x is a ratio of said phase-uncorrected I and Q data.

22. The receiver according to claim 21, wherein said pipelined circuit comprises:

a constant coefficient multiplier; and

a multiplexer for selecting one of a plurality of constant coefficients of said series, an output of said multiplexer being connected to an input of said constant coefficient multiplier.

23. The receiver according to claim 21, wherein said pipelined circuit comprises:

a multiplier;

a first memory for storing the quantity x^2, said first memory being coupled to a first input of said multiplier;

a second memory for holding an output of said multiplier; and

a feedback connection between said second memory and a second input of said multiplier.

24. The receiver according to claim 21, wherein said pipelined circuit further comprises:

a third memory for storing a value of said series;

a control circuit, coupled to said third memory, wherein said pipeline circuit computes N terms of said series, and said pipeline circuit computes N+1 terms of said series, wherein N is an integer;

an averaging circuit coupled to said third memory for computing an average of said N terms and said N+1 terms of said series.

25. The receiver according to claim 1, wherein data transmitted in a pilot carrier of said multicarrier signal is BCH encoded according to a code generator polynomial h(x), further comprising:

a demodulator operative on said BCH encoded data;

an iterative pipelined BCH decoding circuit, comprising:

a circuit coupled to said demodulator for forming a Galois Field of said polynomial, and calculating a plurality of syndromes therewith;

a plurality of storage registers, each said storage register storing a respective one of said syndromes;

a plurality of feedback shift registers, each said feedback shift register accepting data from a respective one of said storage registers and having an output;

a plurality of Galois field multipliers, each said multiplier being connected in a feedback loop across a respective one of said feedback shift registers and multiplying the output of its associated feedback shift register by an alpha value of said Galois Field;

an output Galois field multiplier for multiplying said outputs of two of said feedback shift registers;

an error detection circuit connected to said feedback shift registers and said output Galois field multiplier, wherein an ouput signal of said error detection circuit indicates an error in a current bit of data; and

a feedback line enabled by said error detection circuit and connected to said storage registers, wherein outputs of said feedback shift registers are written into said storage registers.

26. The receiver according to claim 25, wherein said output Galois field multiplier comprises:

a first register initially storing a first multiplicand A;

a constant coefficient multiplier connected to said register for multiplication by a value α, an output of said constant coefficient multiplier being connected to said first register to define a first feedback loop, whereby in a kth cycle of clocked operation said first register contains a Galois field product $A\alpha^k$;

a second register for storing a second multiplicand B;

an AND gate connected to said second register and to said output of said constant coefficient multiplier;

an adder having a first input connected to an output of said AND gate;

an accumulator connected to a second input of said adder; wherein an output of said adder is connected to said accumulator to define a second feedback loop; whereby a Galois field product AB is output by said adder.

27. A method for estimation of a frequency response of a channel, comprising the steps of:

receiving from a channel a multicarrier signal having a plurality of data carriers and scattered pilot carriers, said scattered pilot carriers being spaced apart at a first interval N and being transmitted at a power that differs from a transmitted power of said data carriers;

converting said multicarrier signal to a digital representation thereof;

performing a Fourier transform on said digital representation of said multicarrier signal to generate a transformed digital signal;

reversing a bit order of said transformed digital signal to generate a bit-order reversed signal;

cyclically accumulating magnitudes of carriers in said bit-order reversed signal in N accumulators;

correlating said accumulated magnitudes with said power of said scattered pilot carriers;

responsive to said step of correlating, generating a synchronizing signal that identifies a carrier of said multicarrier signal.

28. The method according to claim 27, wherein said step of accumulating magnitudes comprises the steps of:

adding absolute values of a real component of said bit-order reversed signal to respective absolute values of imaginary components thereof to generate sums;

respectively storing said sums in said accumulators.

29. The method according to claim 27, wherein said step of correlating said accumulated magnitudes further comprises the step of:

identifying a first accumulator having a highest value stored therein representing a first carrier position.

30. The method according to claim 29, wherein said step of correlating said accumulated magnitudes further comprises the steps of:

identifying a second accumulator having a second highest value stored therein representing a second carrier position; and

determining an interval between said first carrier position and said second carrier position.

31. The method according to claim 27, further comprising the steps of:

comparing a position of a carrier of a first symbol in said bit-order reversed signal with a position of a carrier of a second symbol therein.

32. The method according to claim 27, further comprising the steps of:

interpolating between pilot carriers to determine correction factors for respective intermediate data carriers disposed therebetween; and

respectively adjusting magnitudes of said intermediate data carriers according to said correction factors.

33. The method according to claim 27, further comprising the steps of:

determining a mean phase difference between corresponding pilot carriers of successive symbols being transmitted in said transformed digital signal; and

generating a first control signal responsive to said mean phase difference; and

responsive to said first control signal adjusting a frequency of reception of said multicarrier signal.

34. The method according to claim 33, further comprising the steps of:

determining a first phase difference between a first data carrier of a first symbol in said transmitted data carrier and said first data carrier of a second symbol therein;

determining a second phase difference between a second data carrier of said first symbol and said second data carrier of said second symbol; and

determining a difference between said first phase difference and said second phase difference to define a phase slope between said first data carrier and said second data carrier;

generating a second control signal responsive to said phase slope; and

responsive to said second control signal adjusting a sampling frequency of said multicarrier signal.

35. The method according to claim 34, wherein said step of determining a difference between said first phase difference and said second phase difference comprises computing a line of best fit.

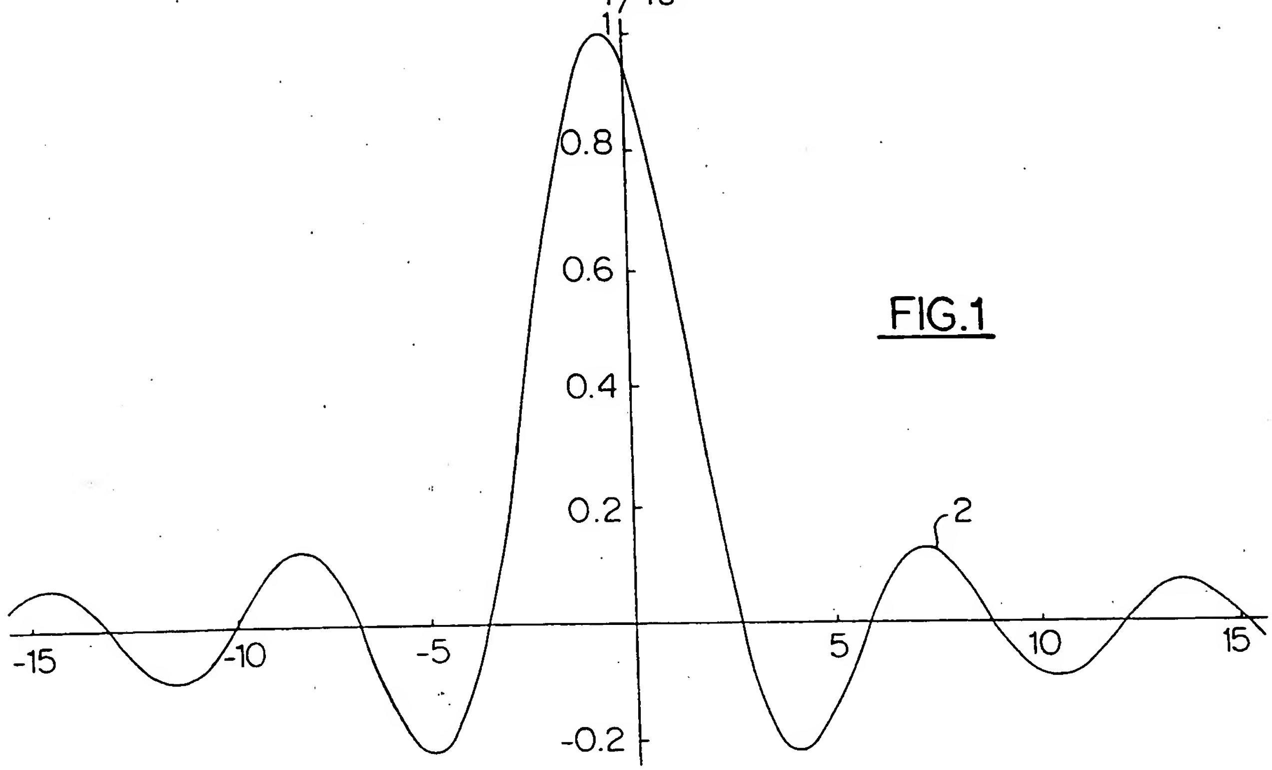


FIG.1

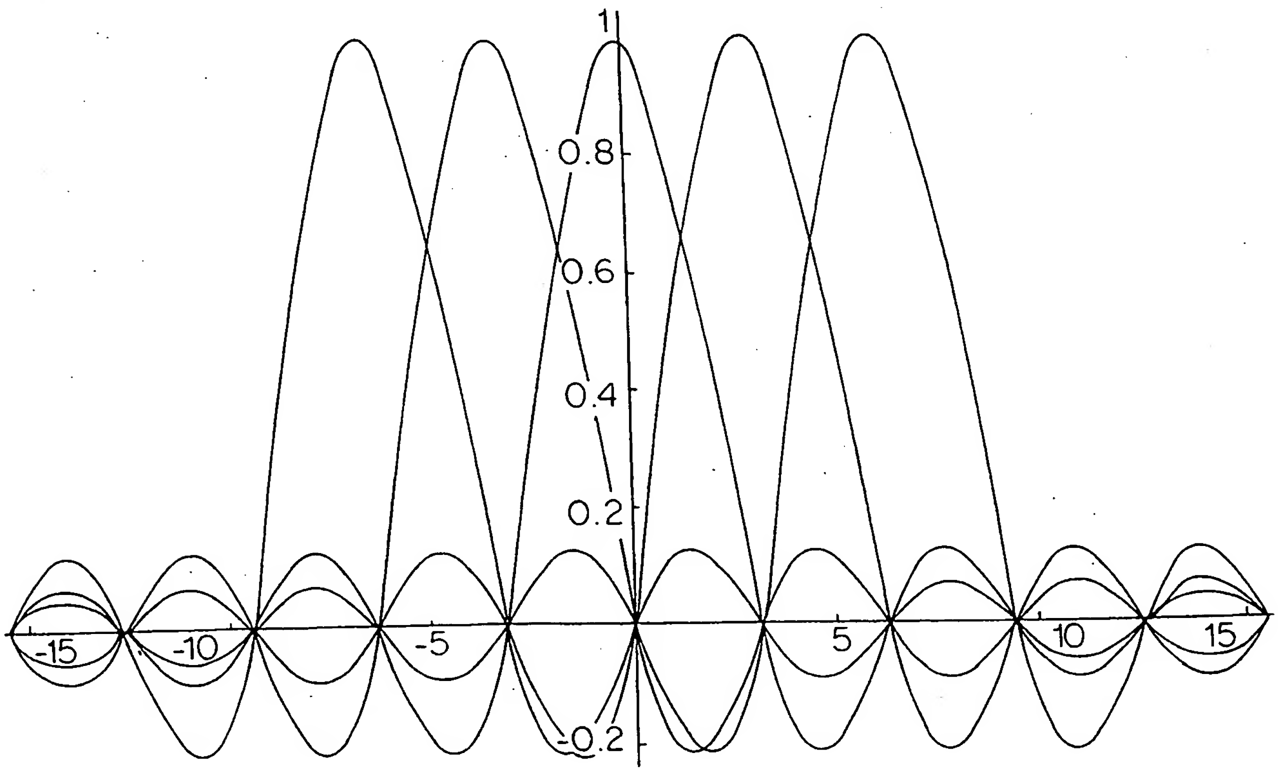


FIG.2

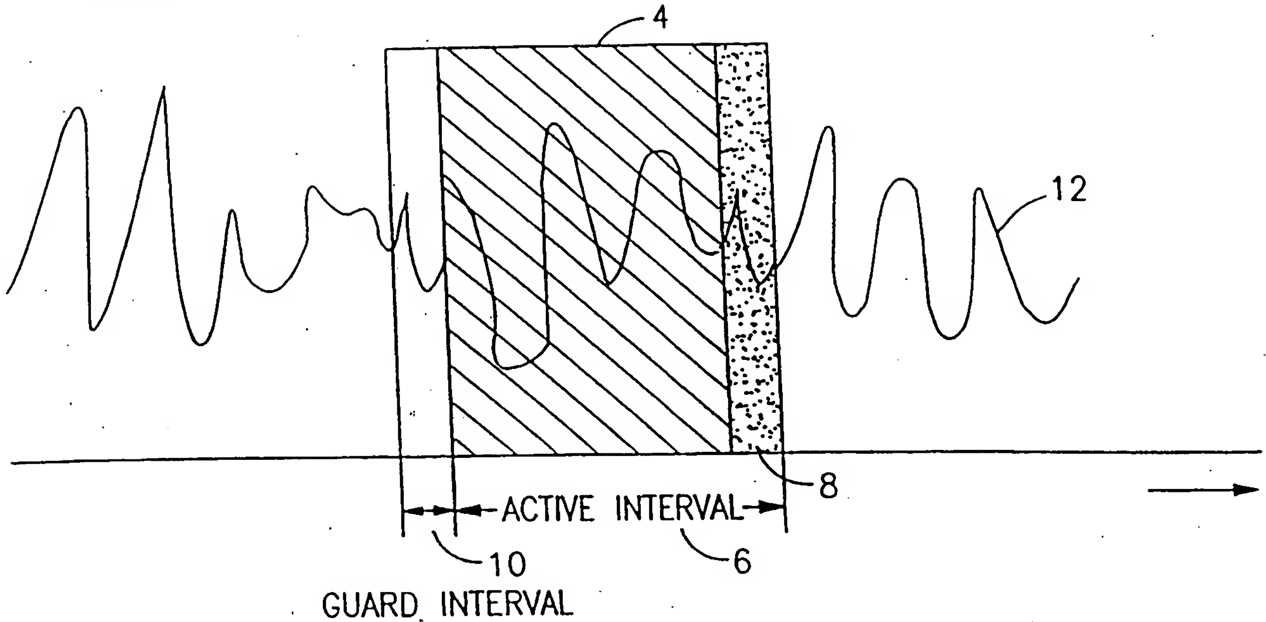

SYMBOL M-1
SYMBOL M
SYMBOL M+1
4
12
8
ACTIVE INTERVAL
10
6
GUARD INTERVAL

FIG.3

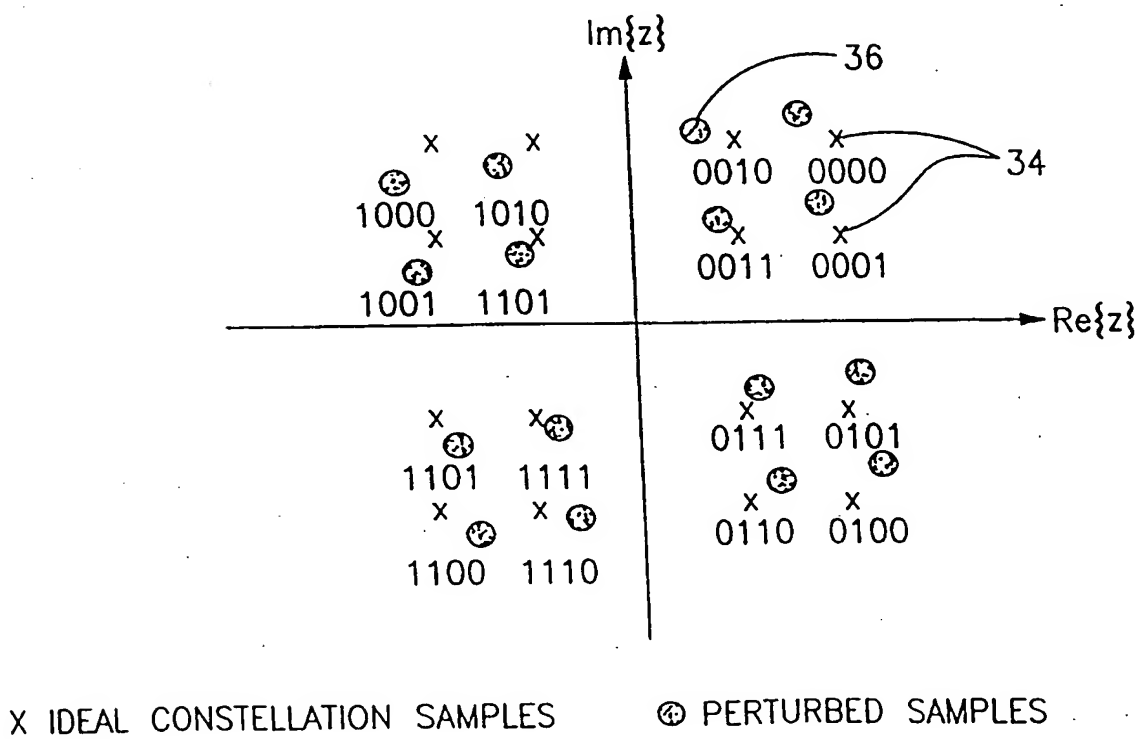

Im{z}
36
34
0010
0000
0011
0001
1000
1010
1001
1101
Re{z}
0111
0101
0110
0100
1101
1111
1100
1110
X IDEAL CONSTELLATION SAMPLES
PERTURBED SAMPLES

FIG.5

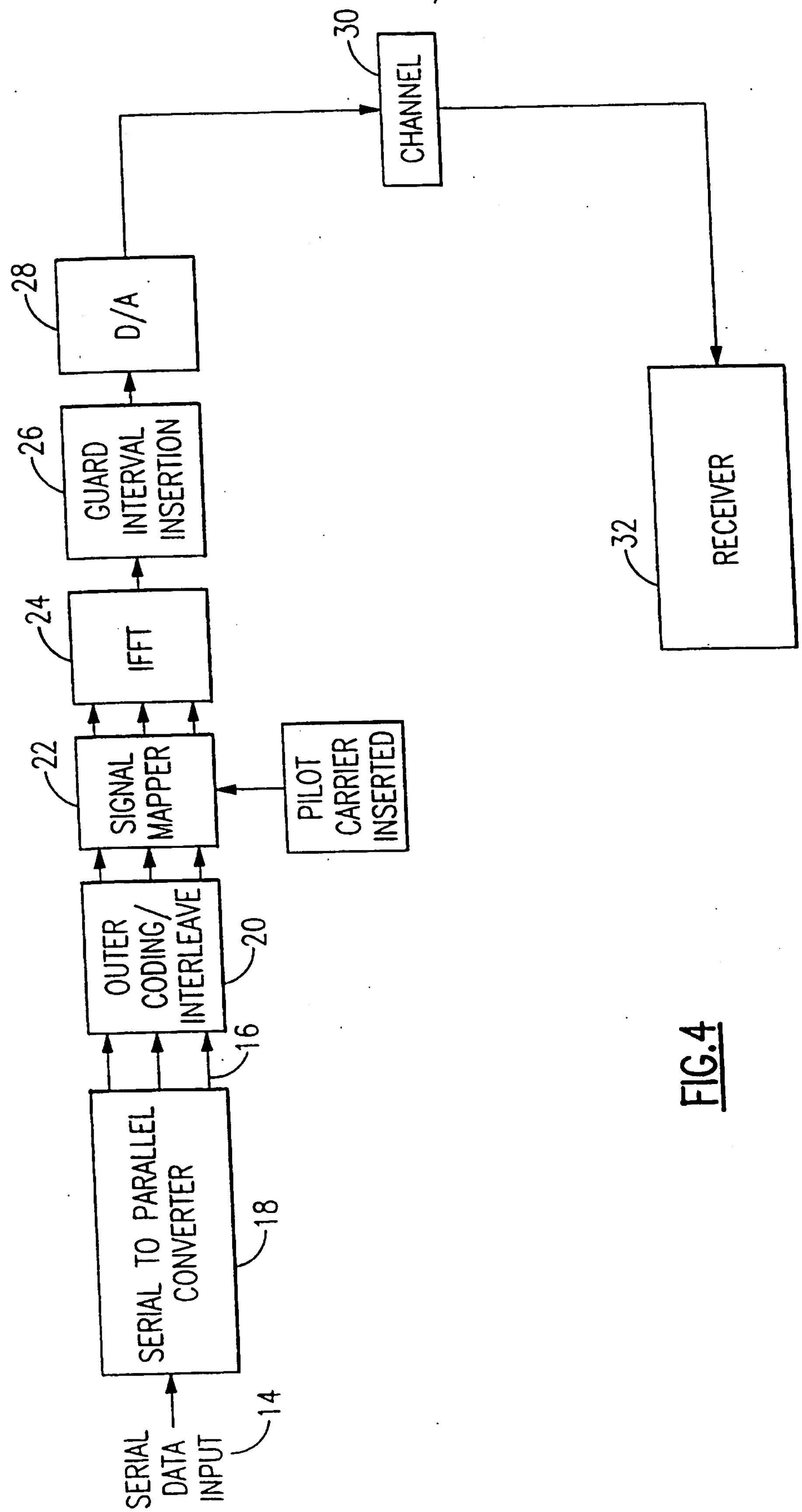

SERIAL DATA INPUT
14
SERIAL TO PARALLEL CONVERTER
18
16
OUTER CODING/ INTERLEAVE
20
SIGNAL MAPPER
22
PILOT CARRIER INSERTED
IFFT
24
GUARD INTERVAL INSERTION
26
D/A
28
CHANNEL
30
RECEIVER
32

FIG.4

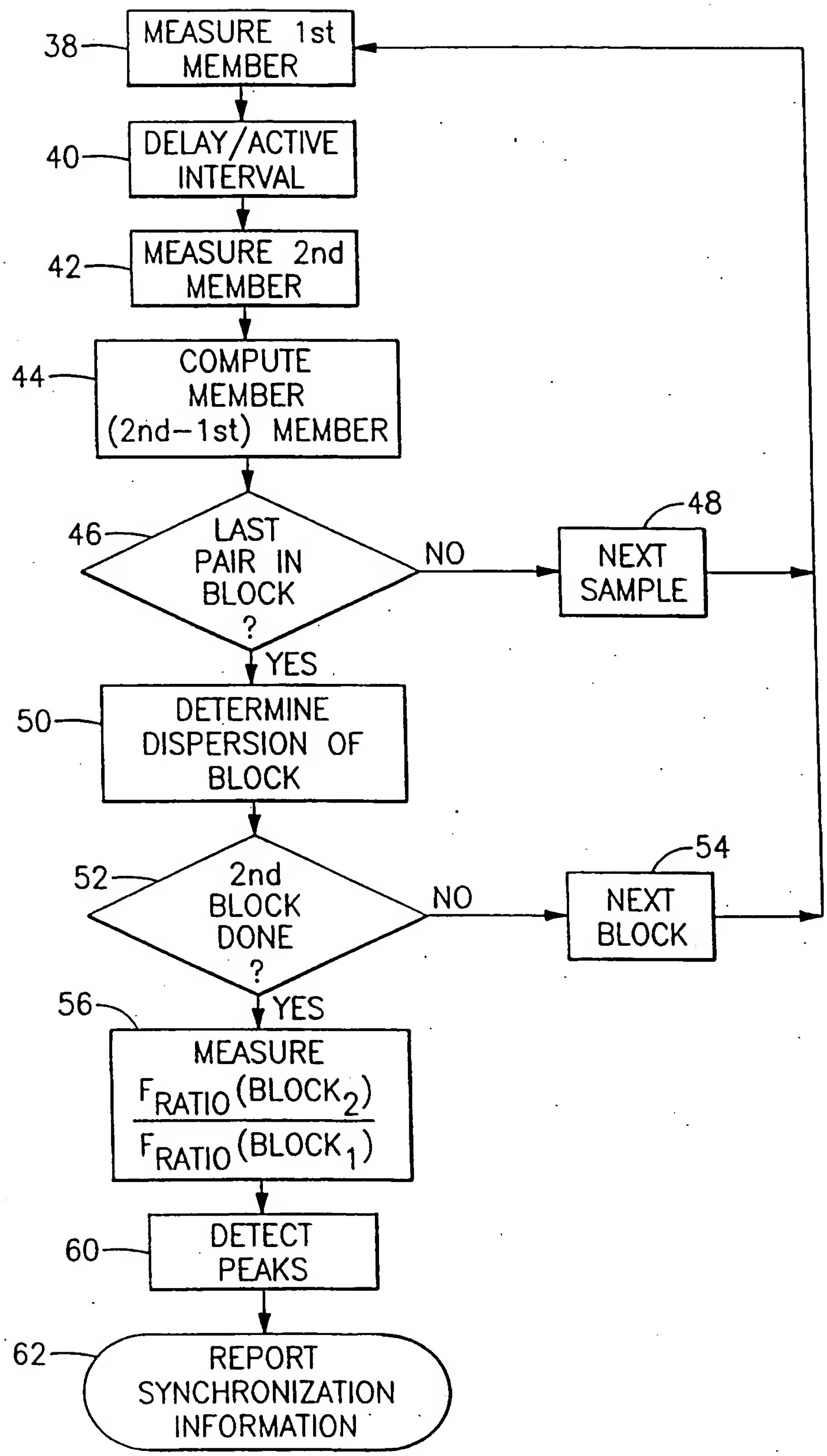

38
MEASURE 1st MEMBER
40
DELAY/ACTIVE INTERVAL
42
MEASURE 2nd MEMBER
44
COMPUTE MEMBER (2nd-1st) MEMBER
46
LAST PAIR IN BLOCK ?
NO
48
NEXT SAMPLE
YES
50
DETERMINE DISPERSION OF BLOCK
52
2nd BLOCK DONE ?
NO
54
NEXT BLOCK
YES
56
MEASURE
F RATIO (BLOCK 2) / F RATIO (BLOCK 1)
60
DETECT PEAKS
62
REPORT SYNCHRONIZATION INFORMATION

FIG.6

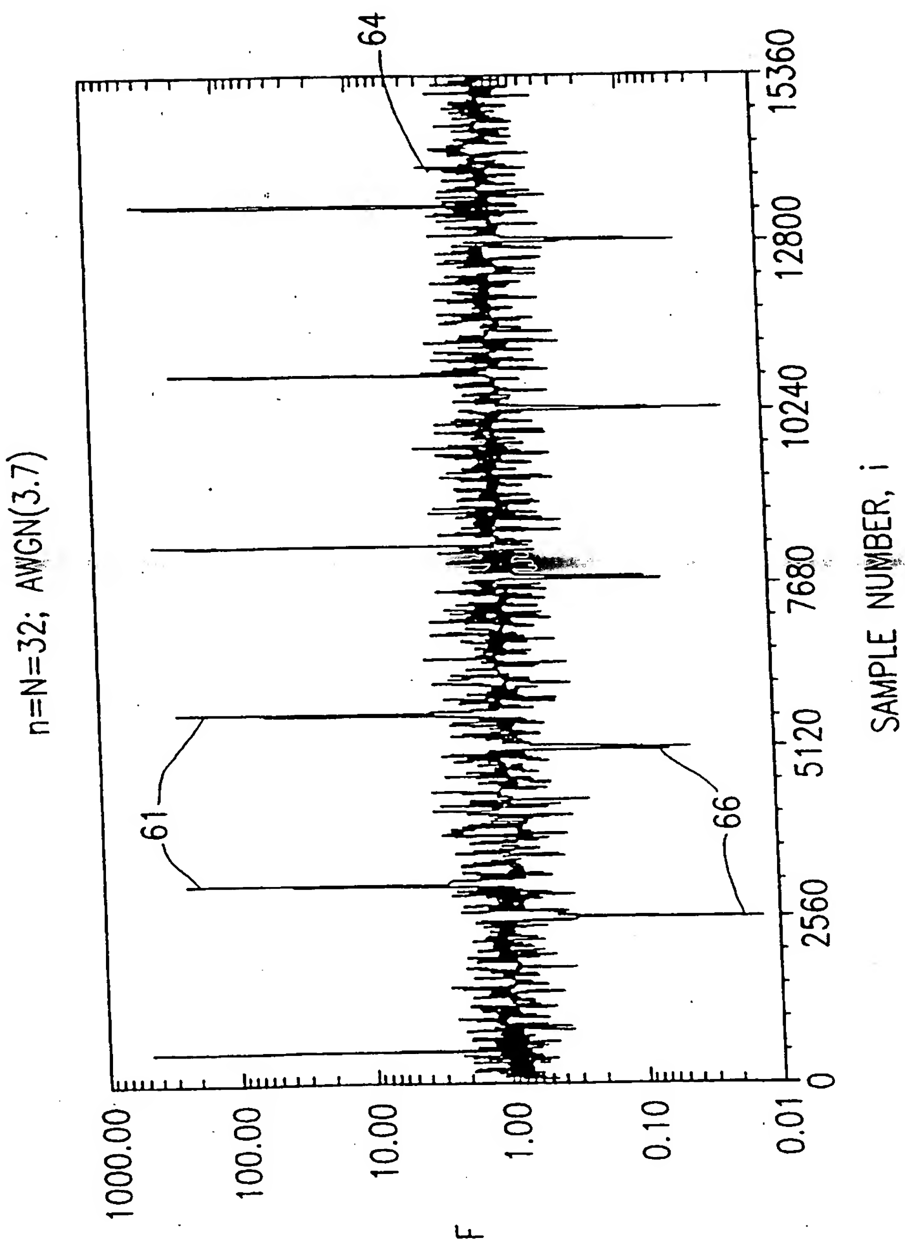

n=N=32; AWGN(3.7)
F
1000.00
100.00
10.00
1.00
0.10
0.01
0
2560
5120
7680
10240
12800
15360
SAMPLE NUMBER, i
61
64
66

FIG.7

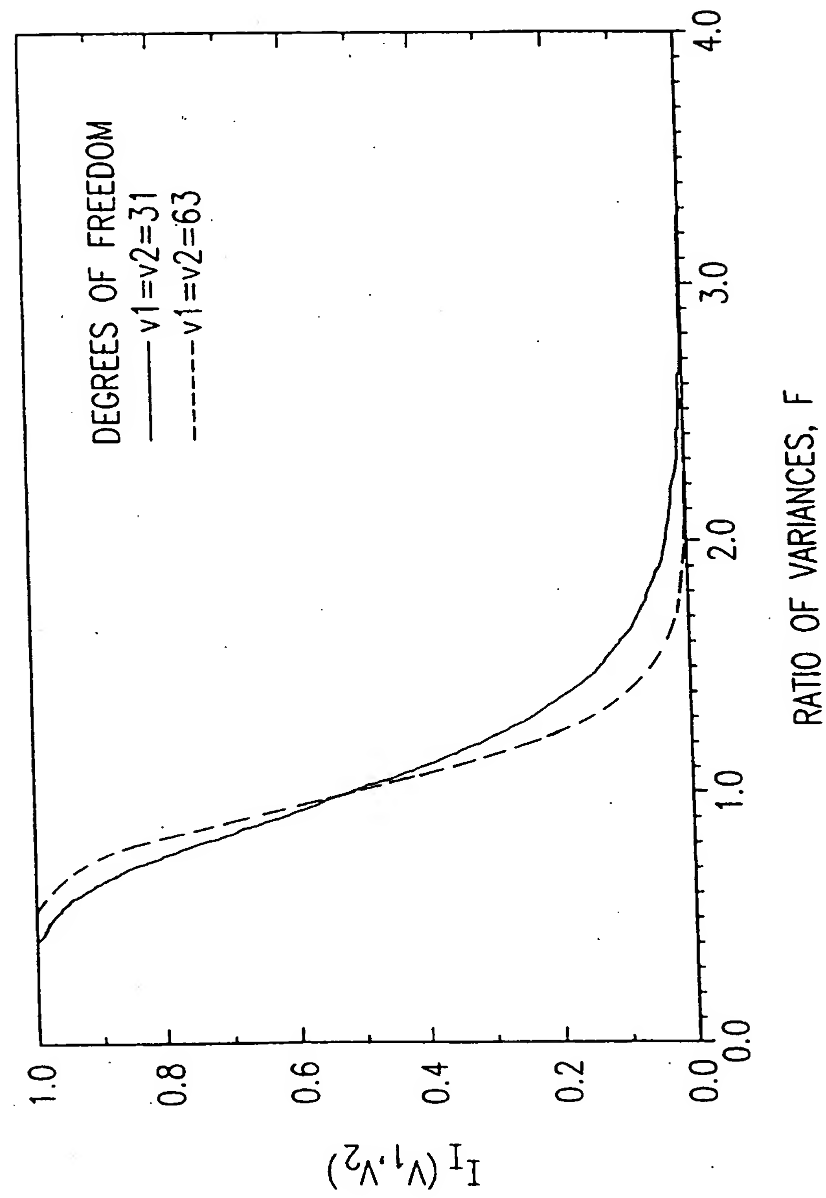

DEGREES OF FREEDOM
v1=v2=31
v1=v2=63
RATIO OF VARIANCES, F
$I_I(V_1,V_2)$
0.0
1.0
2.0
3.0
4.0
0.0
0.2
0.4
0.6
0.8
1.0

FIG.8

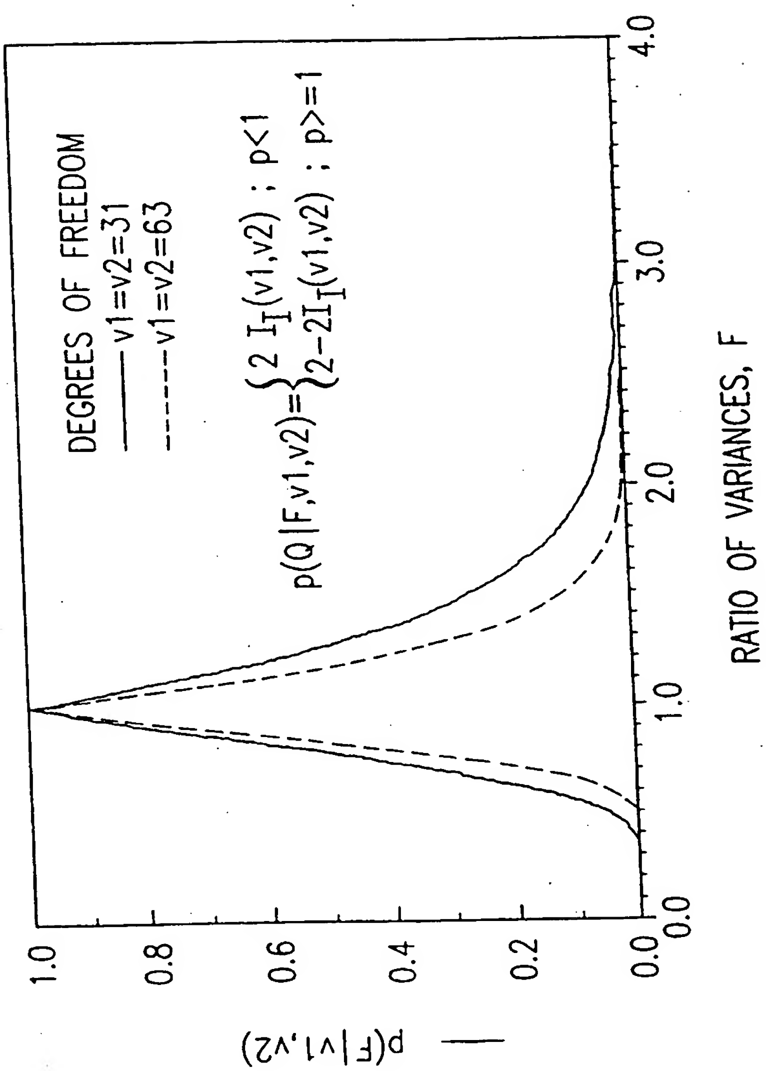
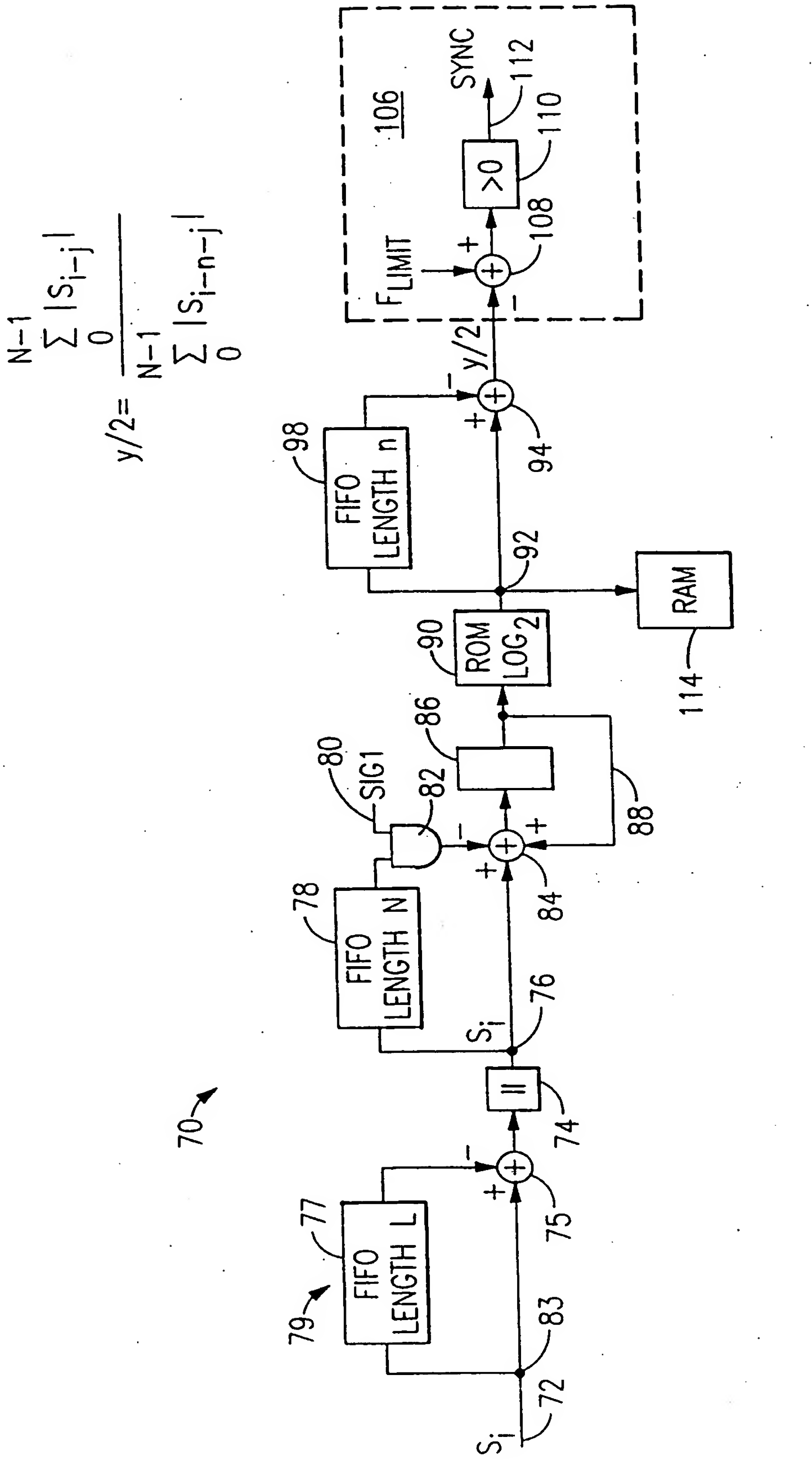

DEGREES OF FREEDOM
v1=v2=31
v1=v2=63
p(Q|F,v1,v2)= 2 I_T(v1,v2) ; p<1
2-2I_T(v1,v2) ; p>=1
RATIO OF VARIANCES, F
p(F|v1,v2)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
1.0
2.0
3.0
4.0

FIG.9



S_i
72
83
79
FIFO LENGTH L
77
75
74
70
S_i
76
FIFO LENGTH N
78
SIG1
80
82
84
86
88
ROM LOG_2
90
92
RAM
114
FIFO LENGTH n
98
94
y/2
F_{LIMIT}
108
>0
110
SYNC
112
106
$$y/2 = \frac{\sum_{0}^{N-1} |S_{i-j}|}{\sum_{0}^{N-1} |S_{i-n-j}|}$$

FIG.10

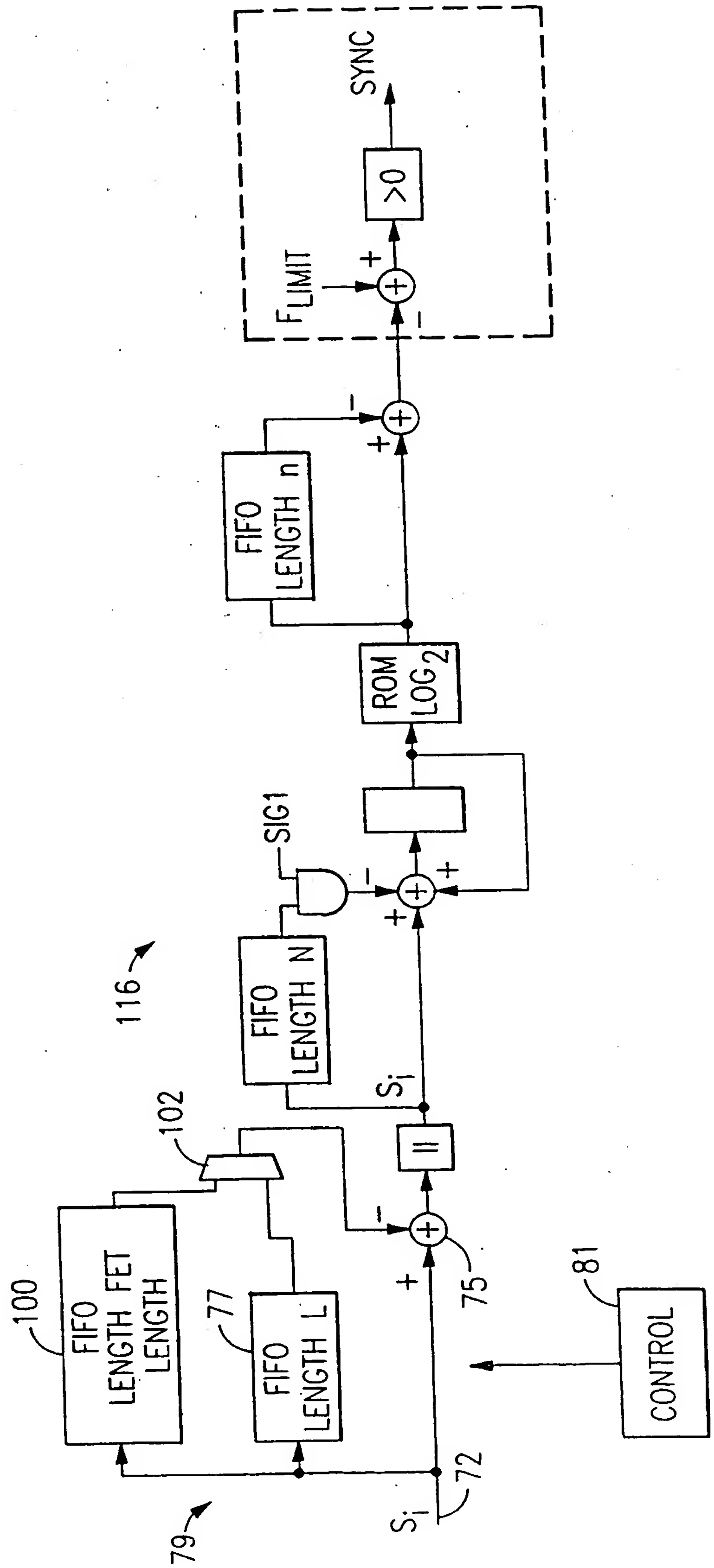
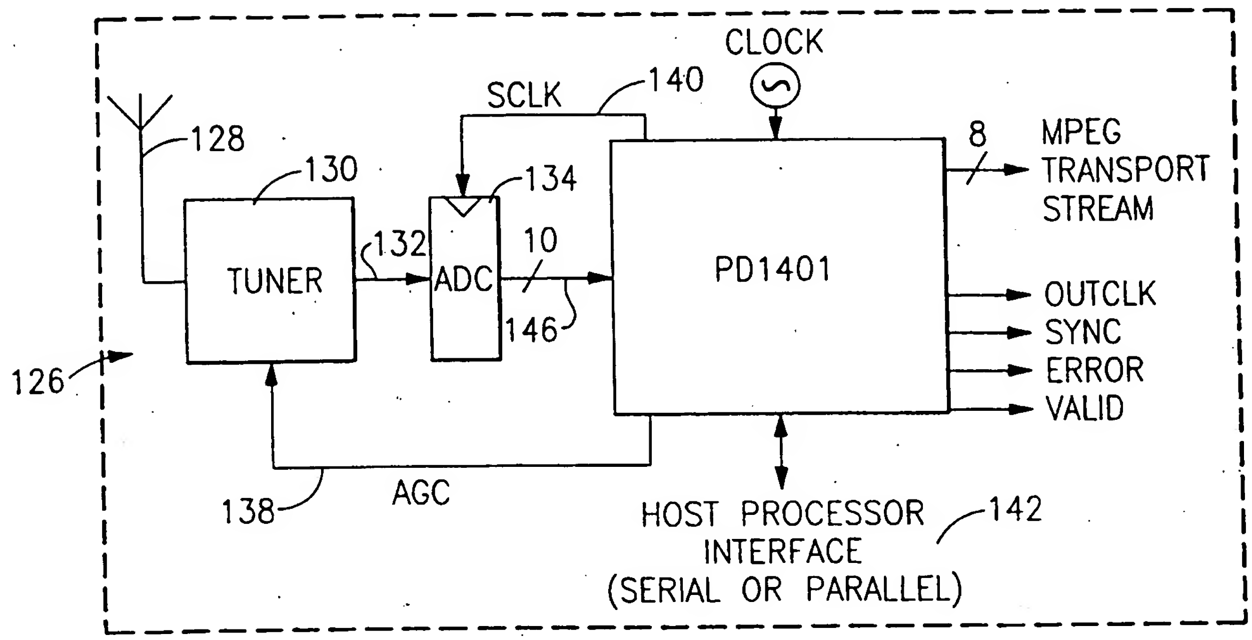
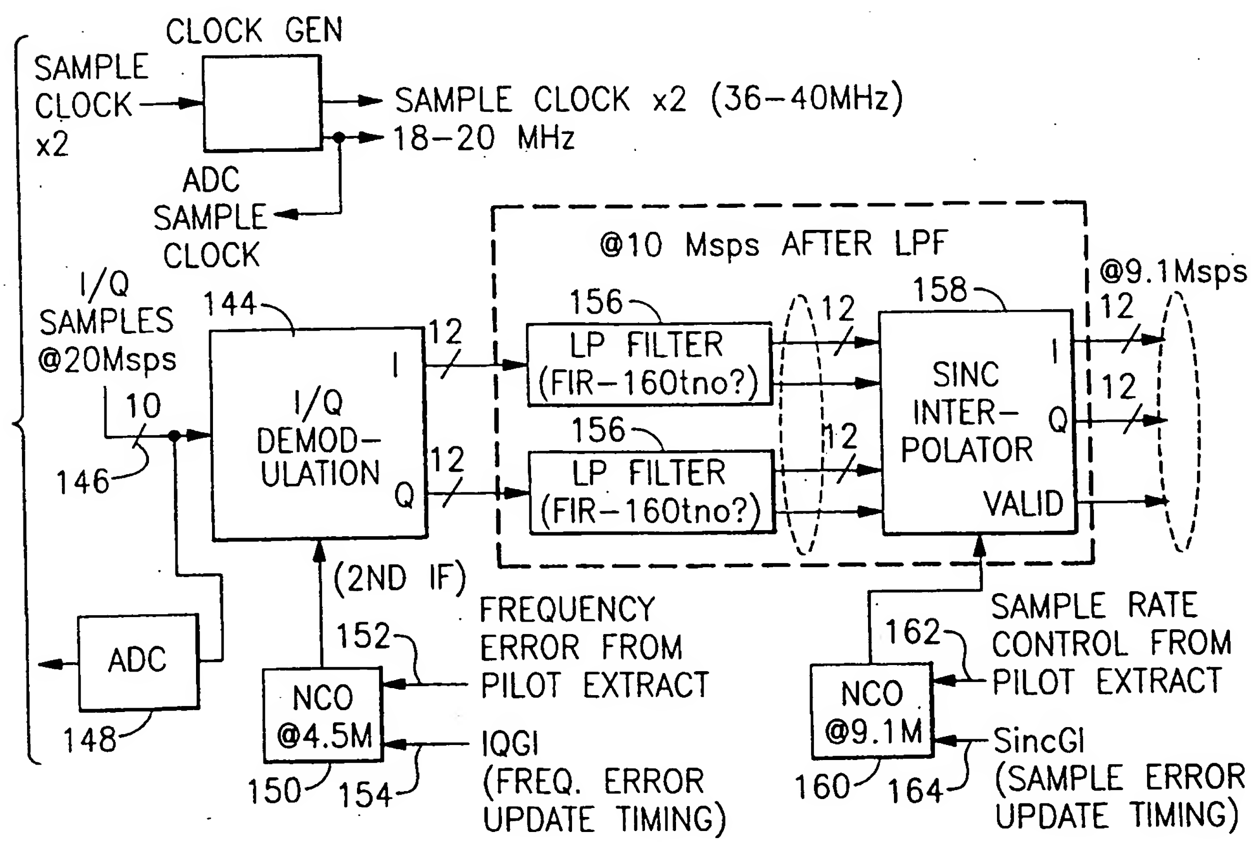

SYNC
>0
F_LIMIT
+
-
FIFO LENGTH n
ROM LOG2
SIG1
116
FIFO LENGTH N
S_i
102
100
FIFO LENGTH FET LENGTH
77
FIFO LENGTH L
75
81
CONTROL
S_i
72
79

FIG.11



CLOCK
SCLK
140
128
130
134
132
10
8
MPEG TRANSPORT STREAM
TUNER
ADC
PD1401
146
OUTCLK
SYNC
ERROR
VALID
126
AGC
138
HOST PROCESSOR INTERFACE (SERIAL OR PARALLEL)
142

FIG.12



CLOCK GEN
SAMPLE CLOCK x2
SAMPLE CLOCK x2 (36-40MHz)
18-20 MHz
ADC SAMPLE CLOCK
@10 Msps AFTER LPF
@9.1Msps
I/Q SAMPLES @20Msps
144
156
158
12
LP FILTER (FIR-160tno?)
SINC INTER-POLATOR
I
Q
VALID
I/Q DEMOD-ULATION
10
146
ADC
148
(2ND IF)
152
FREQUENCY ERROR FROM PILOT EXTRACT
NCO @4.5M
IQGI (FREQ. ERROR UPDATE TIMING)
150
154
162
SAMPLE RATE CONTROL FROM PILOT EXTRACT
NCO @9.1M
SincGI (SAMPLE ERROR UPDATE TIMING)
160
164

FIG.13

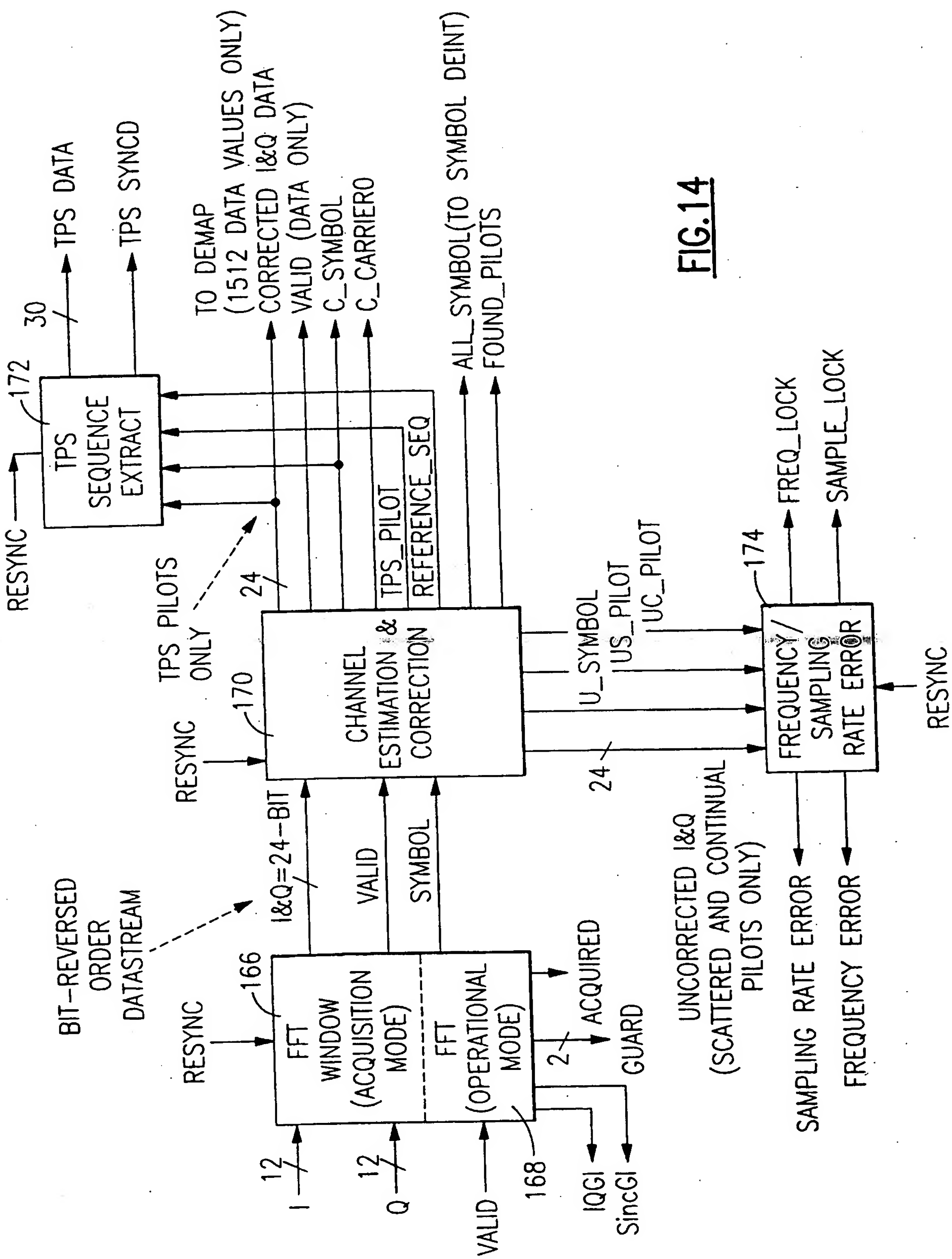

TPS DATA
TPS SYNCD
30
172
TPS SEQUENCE EXTRACT
RESYNC
TO DEMAP
(1512 DATA VALUES ONLY)
CORRECTED I&Q DATA
VALID (DATA ONLY)
C_SYMBOL
C_CARRIER0
ALL_SYMBOL(TO SYMBOL DEINT)
FOUND_PILOTS
TPS_PILOT
REFERENCE_SEQ
24
TPS PILOTS ONLY
170
RESYNC
CHANNEL ESTIMATION & CORRECTION
U_SYMBOL
US_PILOT
UC_PILOT
24
174
FREQ_LOCK
SAMPLE_LOCK
FREQUENCY/ SAMPLING RATE ERROR
RESYNC
UNCORRECTED I&Q
(SCATTERED AND CONTINUAL PILOTS ONLY)
SAMPLING RATE ERROR
FREQUENCY ERROR
BIT-REVERSED ORDER DATASTREAM
I&Q=24-BIT
VALID
SYMBOL
166
RESYNC
FFT WINDOW (ACQUISITION MODE)
FFT (OPERATIONAL MODE)
ACQUIRED
2
GUARD
168
12
I
12
Q
VALID
IQGI
SincGI

FIG.14

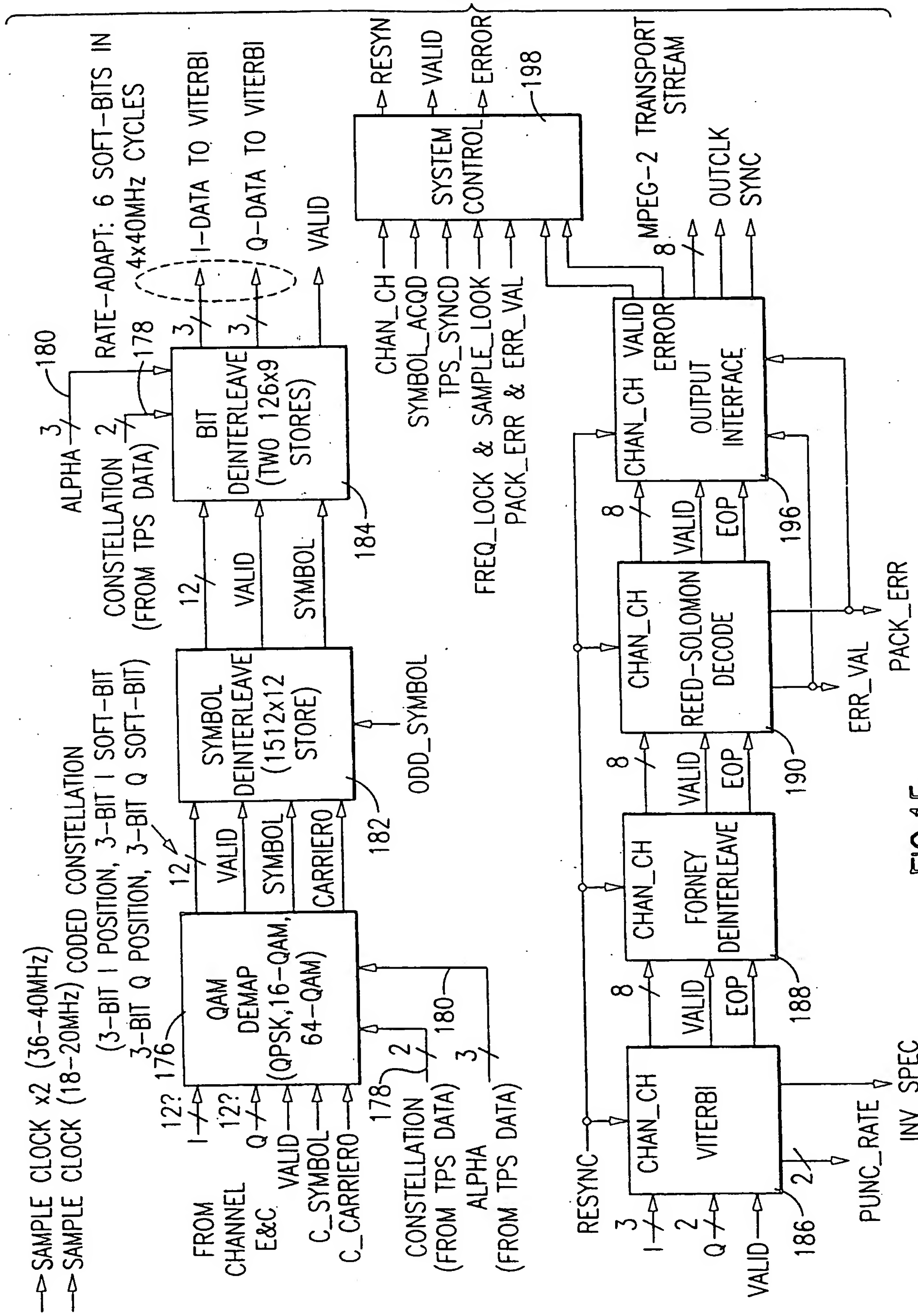

SAMPLE CLOCK x2 (36–40MHz)
SAMPLE CLOCK (18–20MHz)
CODED CONSTELLATION (3-BIT I POSITION, 3-BIT I SOFT-BIT 3-BIT Q POSITION, 3-BIT Q SOFT-BIT)
FROM CHANNEL E&C
I
Q
VALID
C_SYMBOL
C_CARRIER0
12?
12?
176
QAM DEMAP (QPSK,16–QAM, 64–QAM)
178
CONSTELLATION (FROM TPS DATA)
2
ALPHA (FROM TPS DATA)
3
180
12
VALID
SYMBOL
CARRIER0
SYMBOL DEINTERLEAVE (1512x12 STORE)
182
ODD_SYMBOL
12
VALID
SYMBOL
BIT DEINTERLEAVE (TWO 126x9 STORES)
184
ALPHA
3
180
CONSTELLATION (FROM TPS DATA)
2
178
RATE–ADAPT: 6 SOFT–BITS IN 4x40MHz CYCLES
3
3
I–DATA TO VITERBI
Q–DATA TO VITERBI
VALID
CHAN_CH
SYMBOL_ACQD
TPS_SYNCD
FREQ_LOCK & SAMPLE_LOOK
PACK_ERR & ERR_VAL
SYSTEM CONTROL
198
RESYN
VALID
ERROR
RESYNC
I
3
Q
2
VALID
186
CHAN_CH
VITERBI
2
PUNC_RATE
INV_SPEC
8
VALID
EOP
CHAN_CH
FORNEY DEINTERLEAVE
188
8
VALID
EOP
CHAN_CH
REED–SOLOMON DECODE
190
ERR_VAL
PACK_ERR
8
VALID
EOP
CHAN_CH VALID ERROR
OUTPUT INTERFACE
196
8
MPEG–2 TRANSPORT STREAM
OUTCLK
SYNC

FIG.15

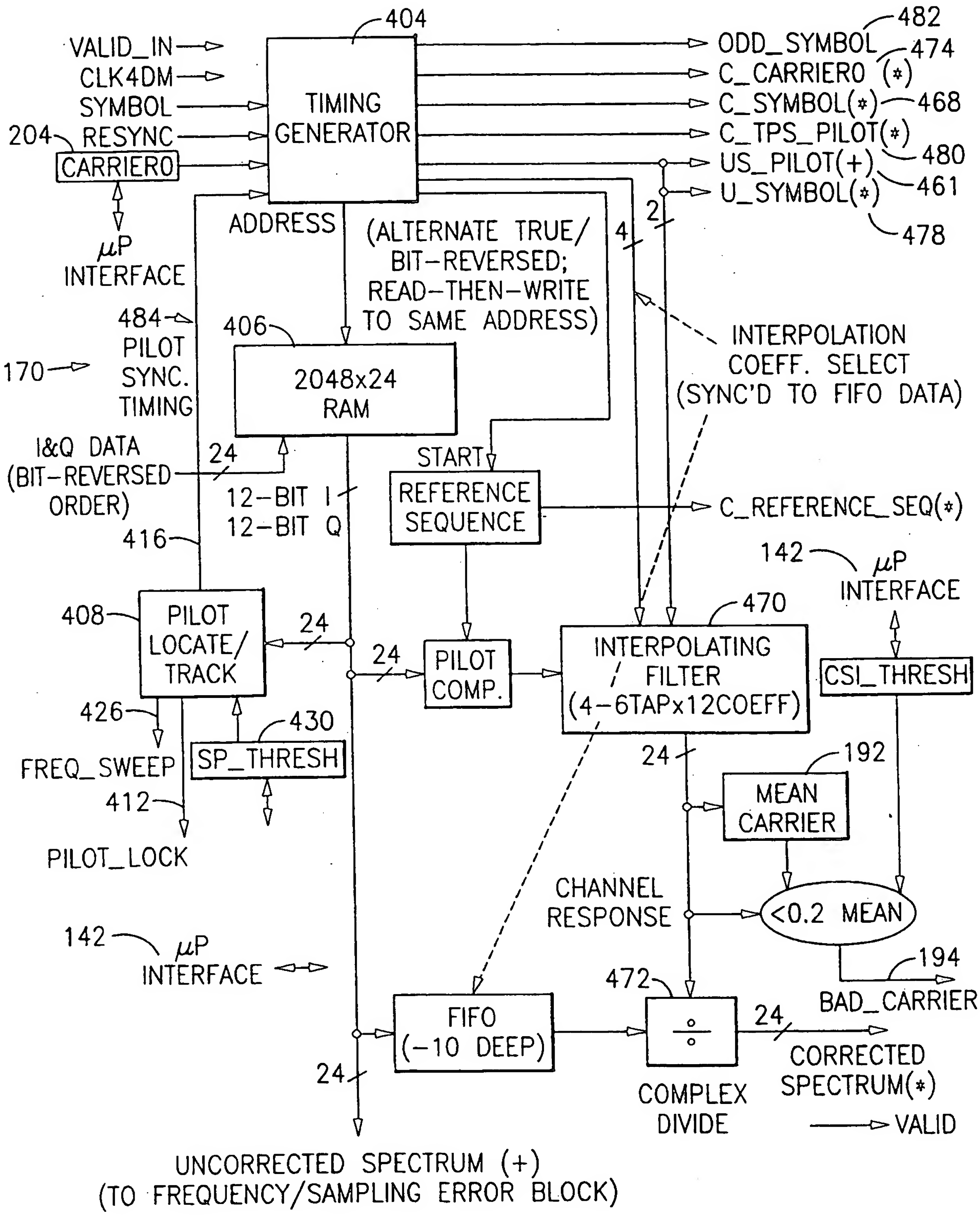

VALID_IN
CLK4DM
SYMBOL
RESYNC
204
CARRIER0
μP INTERFACE
404
TIMING GENERATOR
ADDRESS
(ALTERNATE TRUE/ BIT-REVERSED; READ-THEN-WRITE TO SAME ADDRESS)
ODD_SYMBOL
C_CARRIER0 (*)
C_SYMBOL(*)
C_TPS_PILOT(*)
US_PILOT(+)
U_SYMBOL(*)
482
474
468
480
461
478
2
4
INTERPOLATION COEFF. SELECT (SYNC'D TO FIFO DATA)
484
PILOT SYNC. TIMING
170
406
2048x24 RAM
I&Q DATA (BIT-REVERSED ORDER)
24
12-BIT I
12-BIT Q
START
REFERENCE SEQUENCE
C_REFERENCE_SEQ(*)
416
142
μP INTERFACE
408
PILOT LOCATE/ TRACK
24
24
PILOT COMP.
470
INTERPOLATING FILTER (4-6TAPx12COEFF)
CSI_THRESH
426
430
FREQ_SWEEP
SP_THRESH
412
PILOT_LOCK
24
192
MEAN CARRIER
CHANNEL RESPONSE
<0.2 MEAN
194
BAD_CARRIER
142
μP INTERFACE
472
FIFO (~10 DEEP)
24
COMPLEX DIVIDE
24
CORRECTED SPECTRUM(*)
VALID
UNCORRECTED SPECTRUM (+) (TO FREQUENCY/SAMPLING ERROR BLOCK)

FIG.16

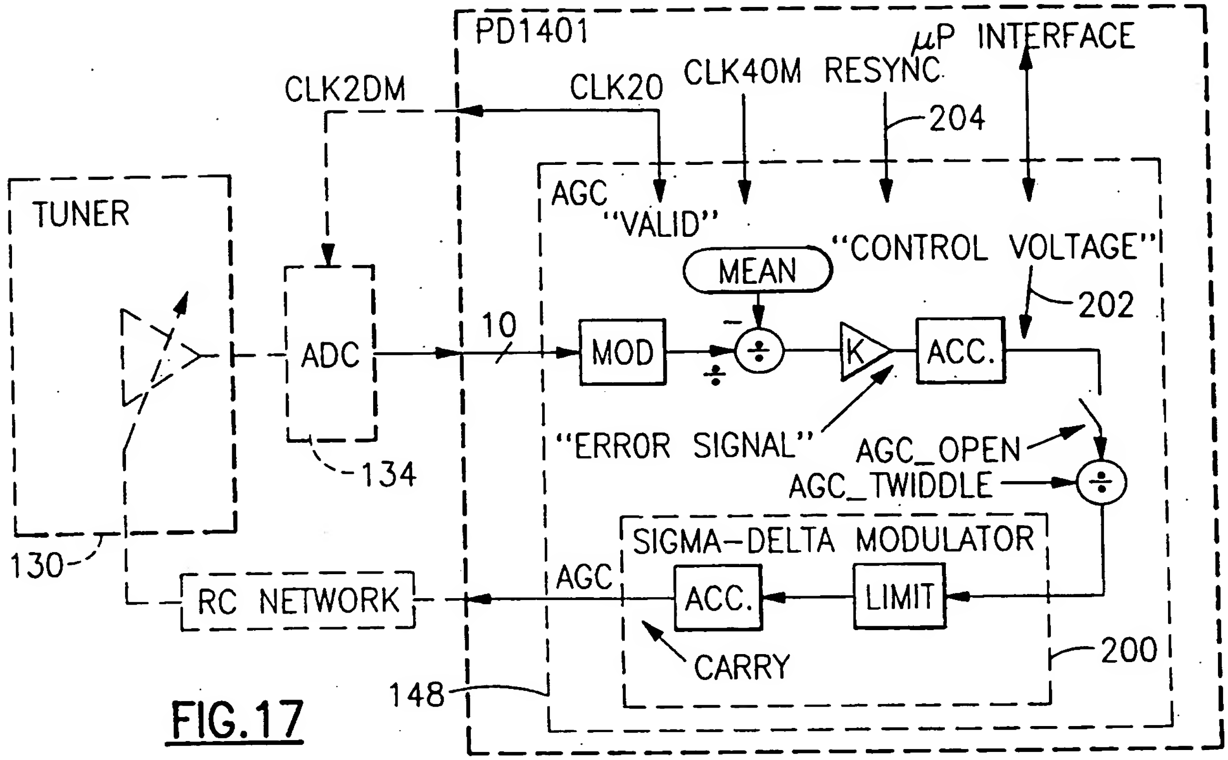

PD1401
μP INTERFACE
CLK2DM
CLK20
CLK40M
RESYNC
204
TUNER
AGC
"VALID"
"CONTROL VOLTAGE"
MEAN
202
ADC
10
MOD
K
ACC.
"ERROR SIGNAL"
AGC_OPEN
AGC_TWIDDLE
134
130
SIGMA-DELTA MODULATOR
RC NETWORK
AGC
ACC.
LIMIT
CARRY
200
148

FIG.17

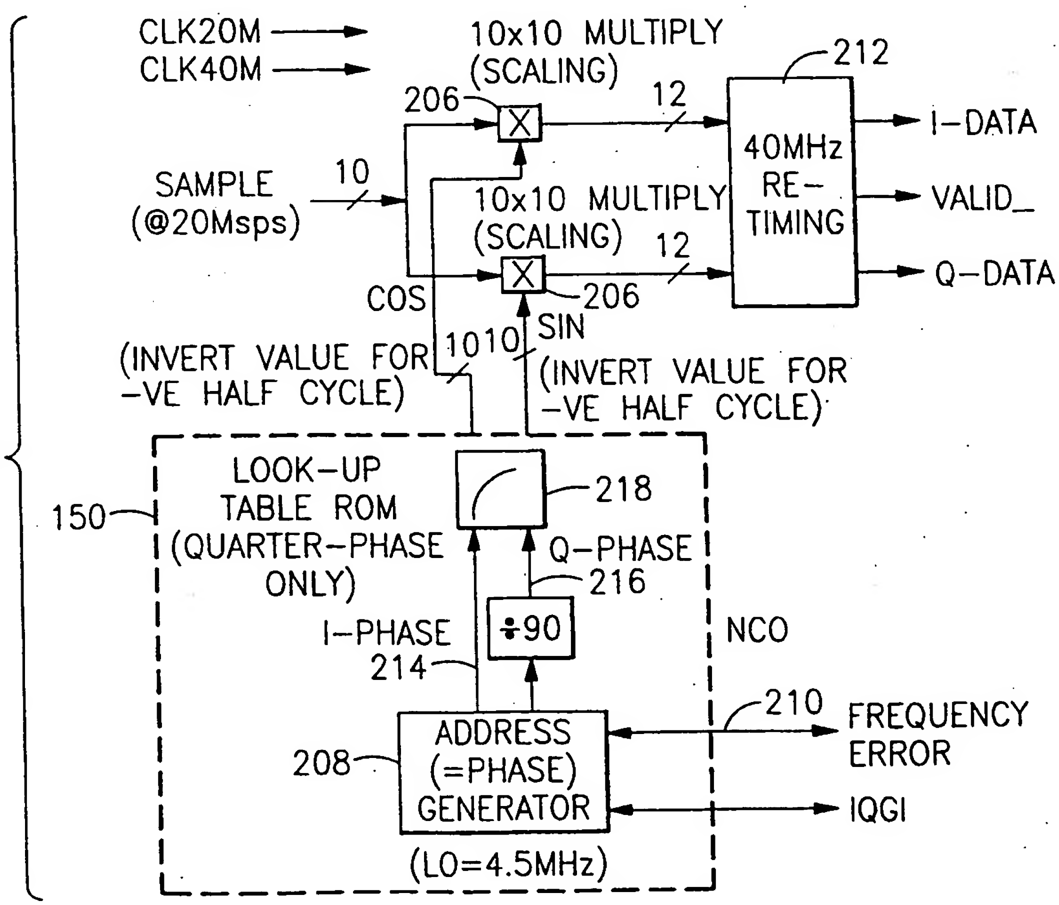

CLK20M
CLK40M
10x10 MULTIPLY
(SCALING)
212
206
12
40MHz
RE-
TIMING
I-DATA
SAMPLE
(@20Msps)
10
10x10 MULTIPLY
(SCALING)
VALID_
12
Q-DATA
COS
206
SIN
(INVERT VALUE FOR
-VE HALF CYCLE)
10
10
(INVERT VALUE FOR
-VE HALF CYCLE)
LOOK-UP
TABLE ROM
(QUARTER-PHASE
ONLY)
218
150
Q-PHASE
216
÷90
NCO
I-PHASE
214
210
FREQUENCY
ERROR
ADDRESS
(=PHASE)
GENERATOR
208
IQGI
(LO=4.5MHz)

FIG.18

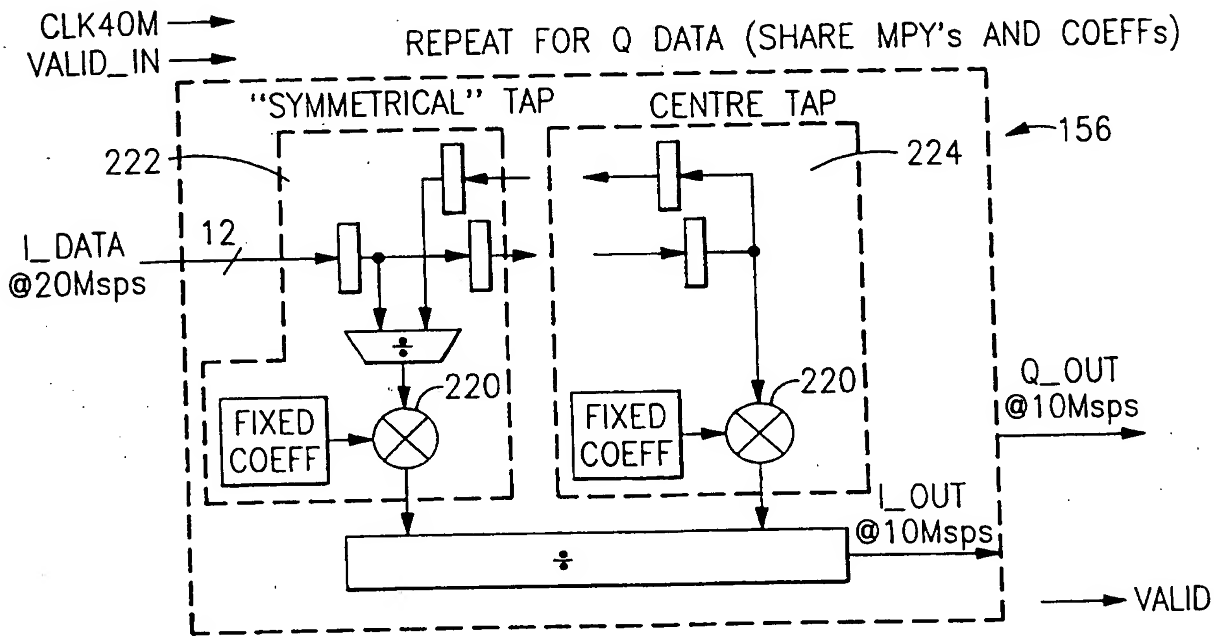

CLK40M
VALID_IN
REPEAT FOR Q DATA (SHARE MPY's AND COEFFs)
"SYMMETRICAL" TAP
CENTRE TAP
156
222
224
I_DATA
@20Msps
12
220
220
FIXED COEFF
FIXED COEFF
Q_OUT
@10Msps
I_OUT
@10Msps
VALID

FIG.19

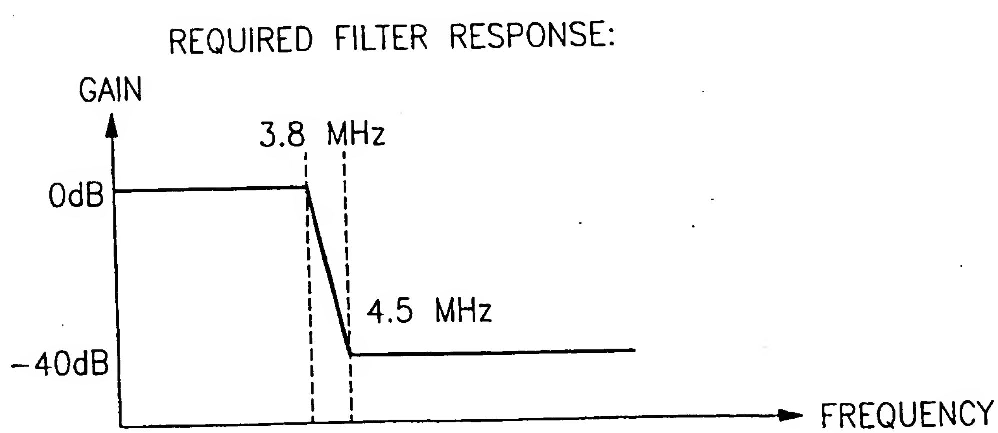

REQUIRED FILTER RESPONSE:
GAIN
3.8 MHz
0dB
4.5 MHz
-40dB
FREQUENCY

FIG.20

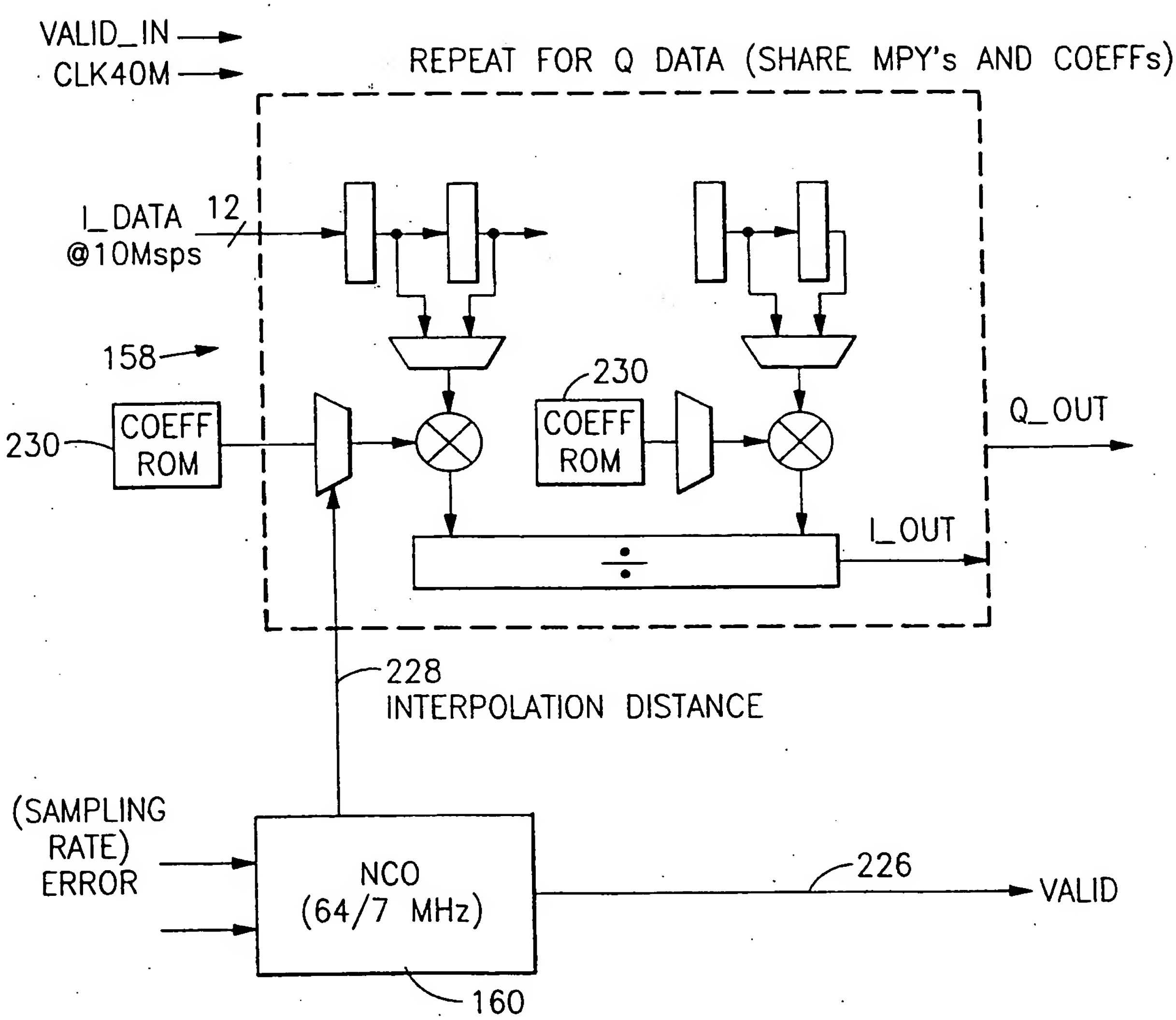

VALID_IN
CLK40M
REPEAT FOR Q DATA (SHARE MPY's AND COEFFs)
I_DATA
@10Msps
12
158
230
COEFF ROM
230
COEFF ROM
Q_OUT
I_OUT
÷
228
INTERPOLATION DISTANCE
(SAMPLING RATE) ERROR
NCO
(64/7 MHz)
226
VALID
160

FIG.21

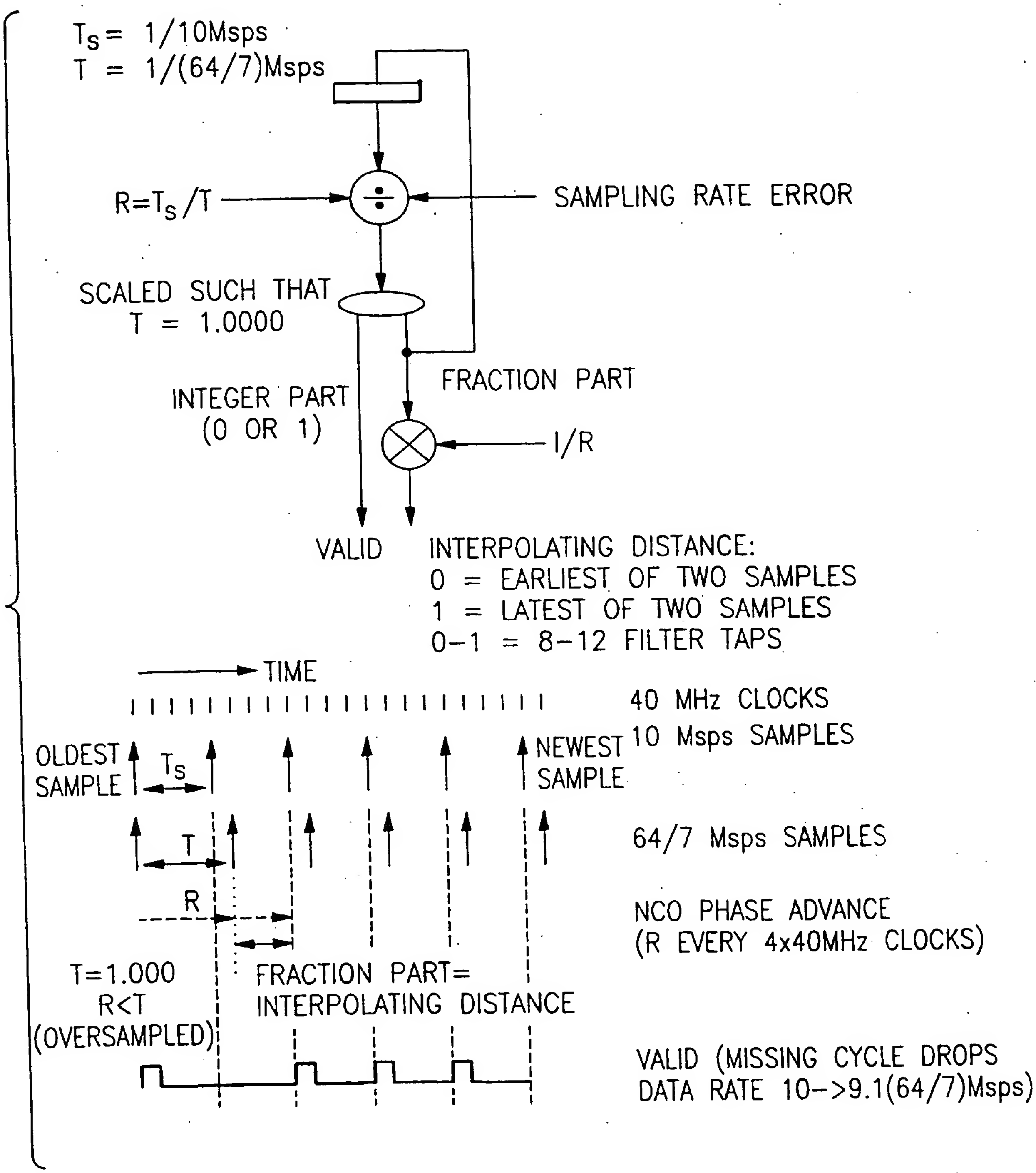
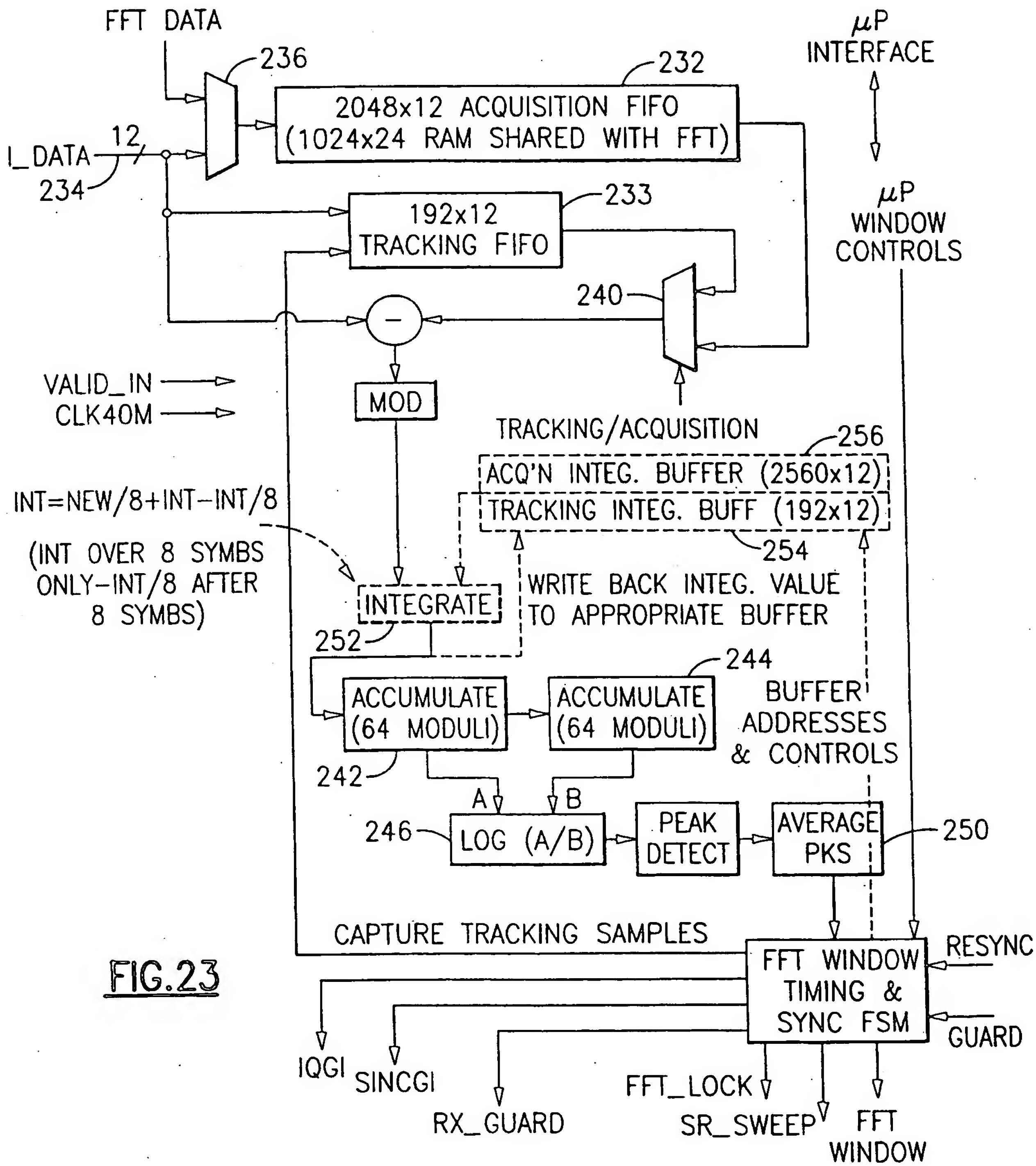

T_S = 1/10Msps
T = 1/(64/7)Msps
R=T_S/T
SAMPLING RATE ERROR
SCALED SUCH THAT
T = 1.0000
FRACTION PART
INTEGER PART
(0 OR 1)
I/R
VALID
INTERPOLATING DISTANCE:
0 = EARLIEST OF TWO SAMPLES
1 = LATEST OF TWO SAMPLES
0-1 = 8-12 FILTER TAPS
TIME
40 MHz CLOCKS
10 Msps SAMPLES
OLDEST
SAMPLE
T_S
NEWEST
SAMPLE
T
64/7 Msps SAMPLES
R
NCO PHASE ADVANCE
(R EVERY 4x40MHz CLOCKS)
T=1.000
R<T
(OVERSAMPLED)
FRACTION PART=
INTERPOLATING DISTANCE
VALID (MISSING CYCLE DROPS
DATA RATE 10->9.1(64/7)Msps)

FIG.22



FFT DATA
236
232
μP INTERFACE
2048x12 ACQUISITION FIFO (1024x24 RAM SHARED WITH FFT)
I_DATA
12
234
192x12 TRACKING FIFO
233
μP WINDOW CONTROLS
240
–
VALID_IN
CLK40M
MOD
TRACKING/ACQUISITION
256
ACQ'N INTEG. BUFFER (2560x12)
TRACKING INTEG. BUFF (192x12)
254
INT=NEW/8+INT–INT/8
(INT OVER 8 SYMBS ONLY–INT/8 AFTER 8 SYMBS)
INTEGRATE
WRITE BACK INTEG. VALUE TO APPROPRIATE BUFFER
252
244
ACCUMULATE (64 MODULI)
ACCUMULATE (64 MODULI)
BUFFER ADDRESSES & CONTROLS
242
A
B
246
LOG (A/B)
PEAK DETECT
AVERAGE PKS
250
CAPTURE TRACKING SAMPLES
FFT WINDOW TIMING & SYNC FSM
RESYNC
GUARD
IQGI
SINCGI
RX_GUARD
FFT_LOCK
SR_SWEEP
FFT WINDOW

FIG.23

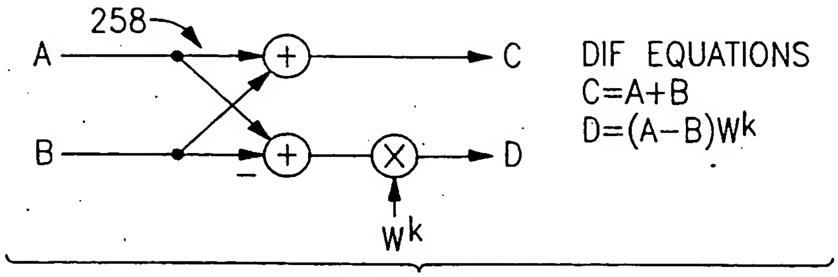


FIG.24

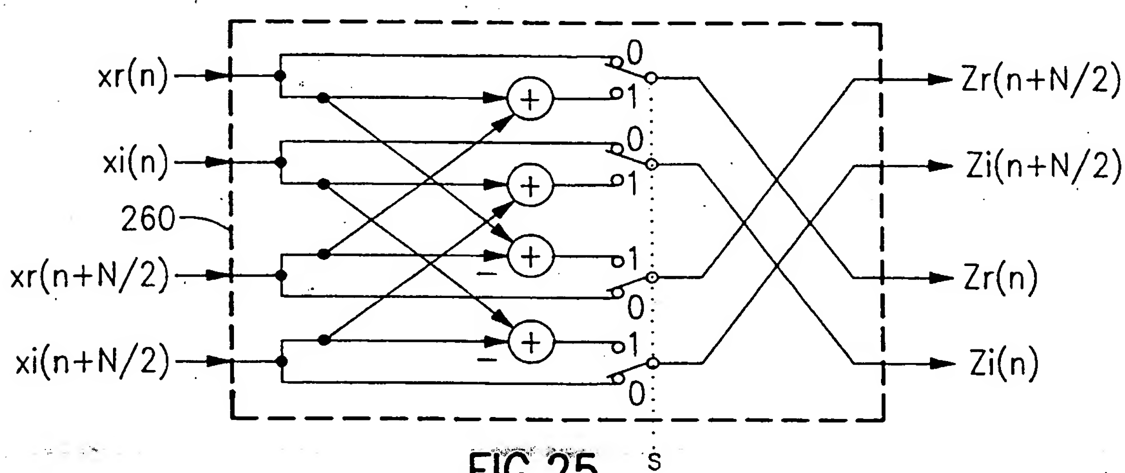


FIG.25
Prior Art

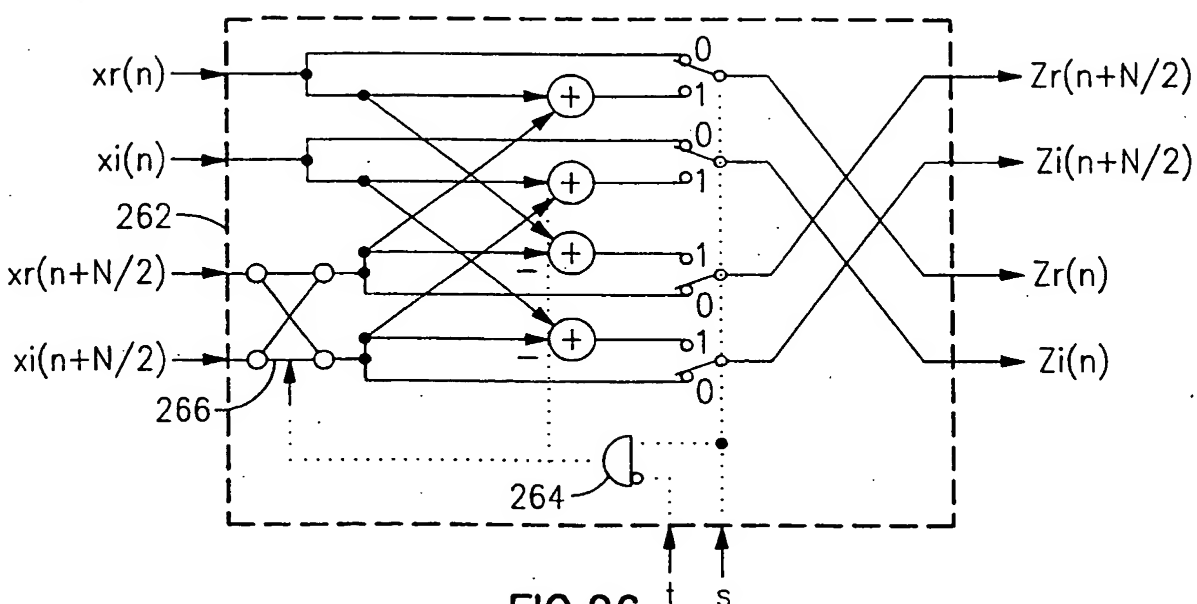


FIG.26
Prior Art

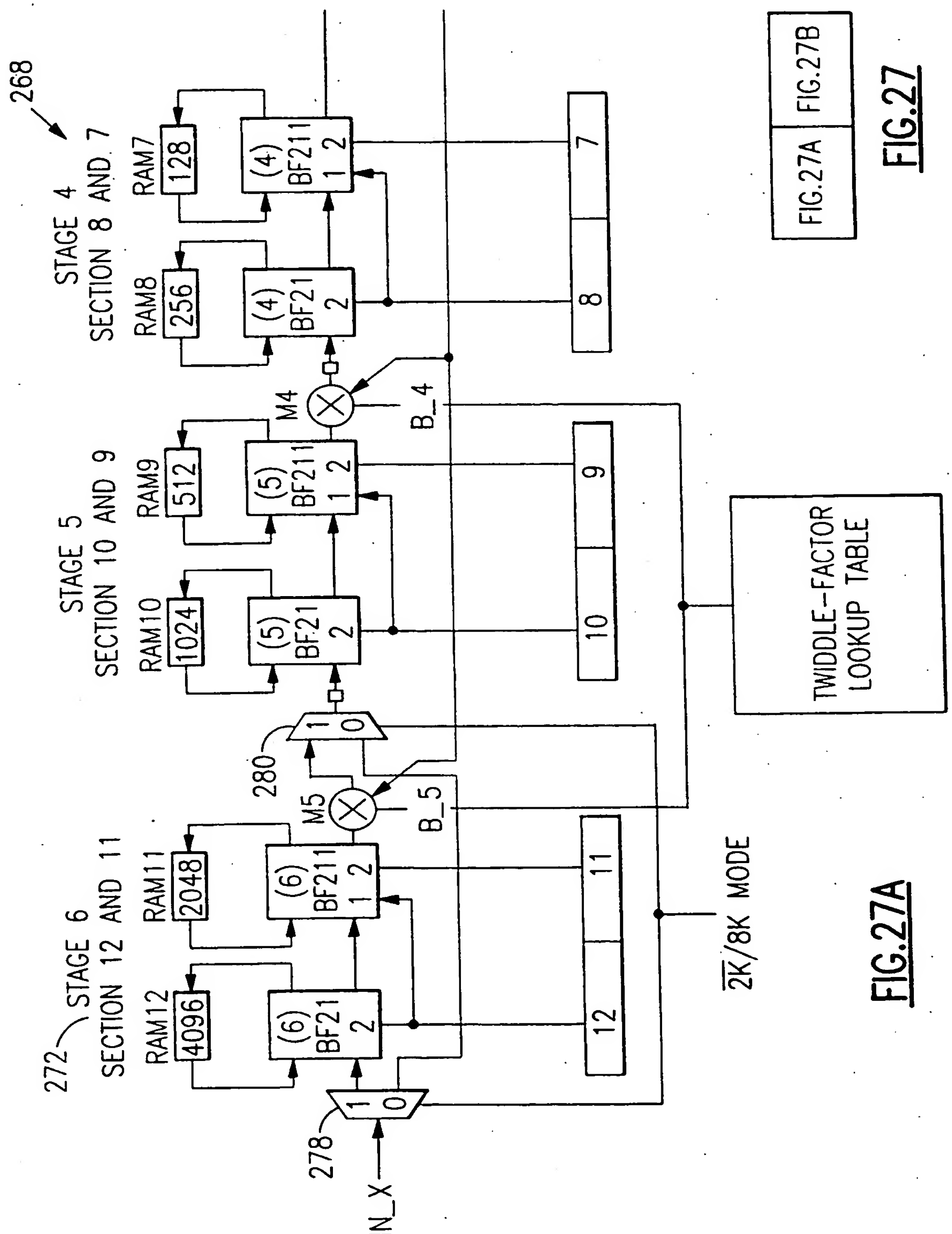


FIG.27A

FIG.27

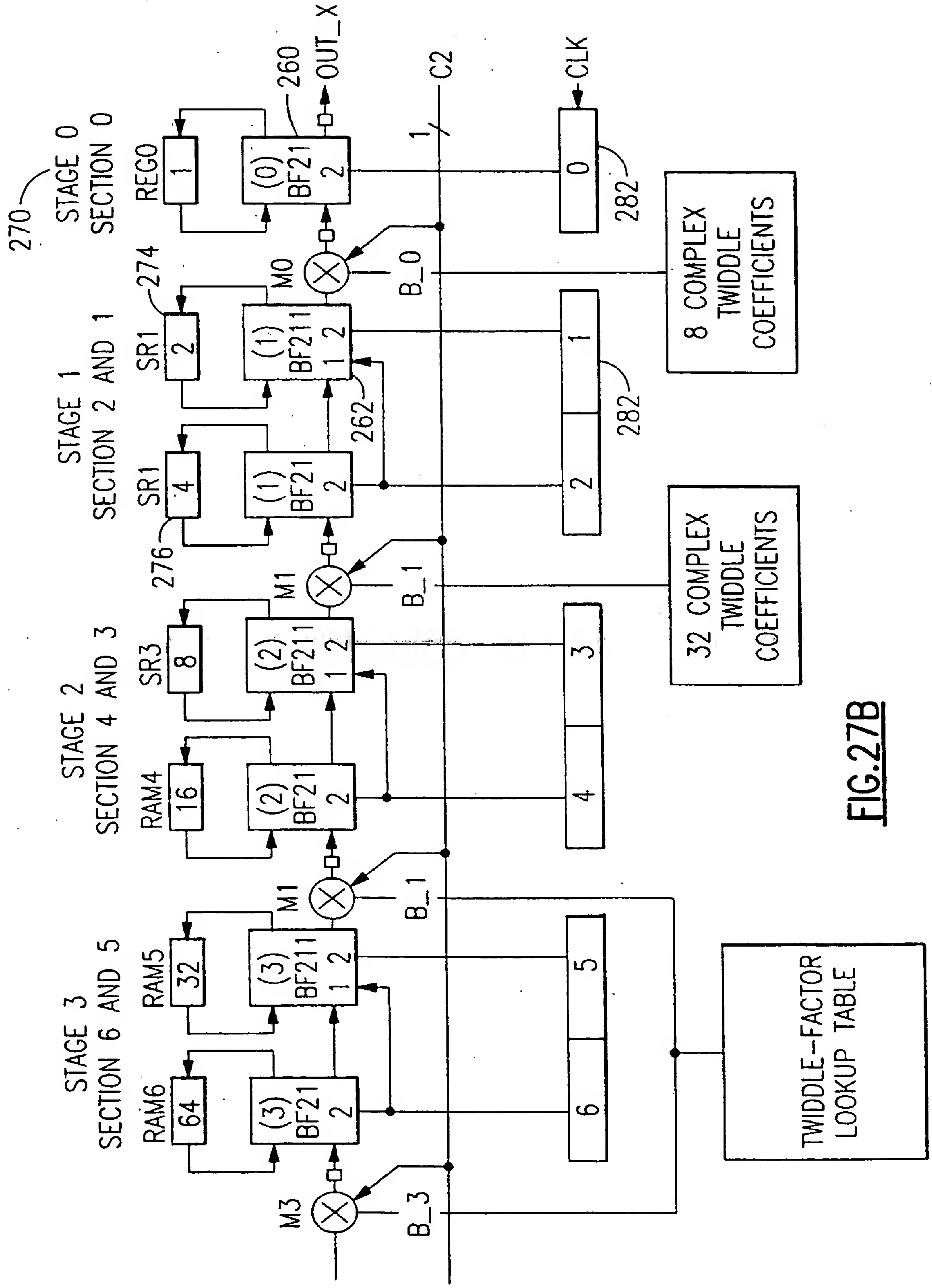

270
STAGE 0
SECTION 0
REG0
1
(0)
BF21
2
260
OUT_X
C2
1
CLK
0
282
M0
B_0
8 COMPLEX
TWIDDLE
COEFFICIENTS
274
STAGE 1
SECTION 2 AND 1
SR1
2
(1)
BF211
1 2
262
1
282
SR1
4
(1)
BF21
2
2
276
M1
B_1
32 COMPLEX
TWIDDLE
COEFFICIENTS
STAGE 2
SECTION 4 AND 3
SR3
8
(2)
BF211
1 2
3
RAM4
16
(2)
BF21
2
4
M1
B_1
STAGE 3
SECTION 6 AND 5
RAM5
32
(3)
BF211
1 2
5
RAM6
64
(3)
BF21
2
6
TWIDDLE-FACTOR
LOOKUP TABLE
M3
B_3

FIG.27B

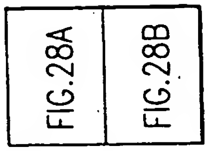


FIG.28

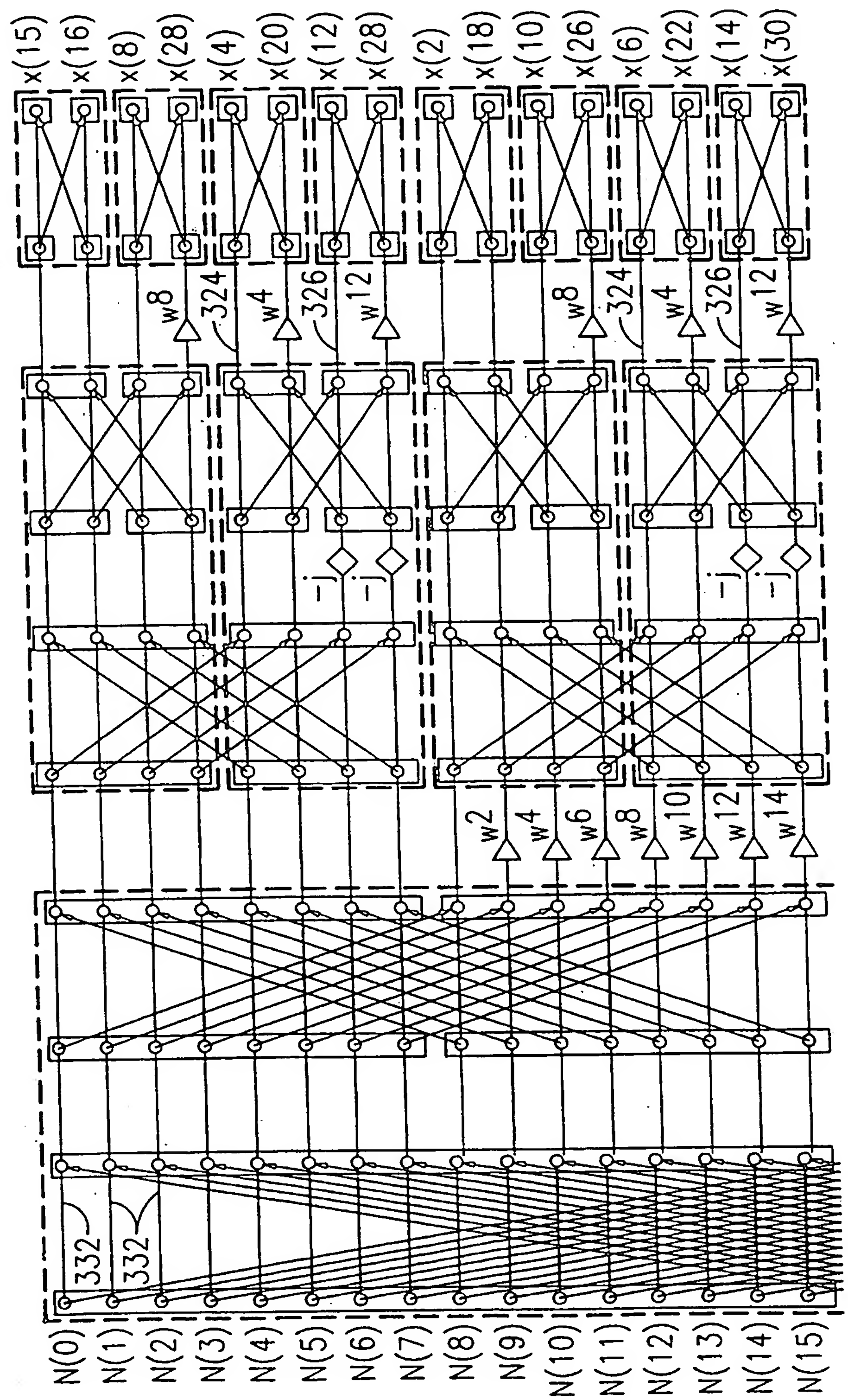
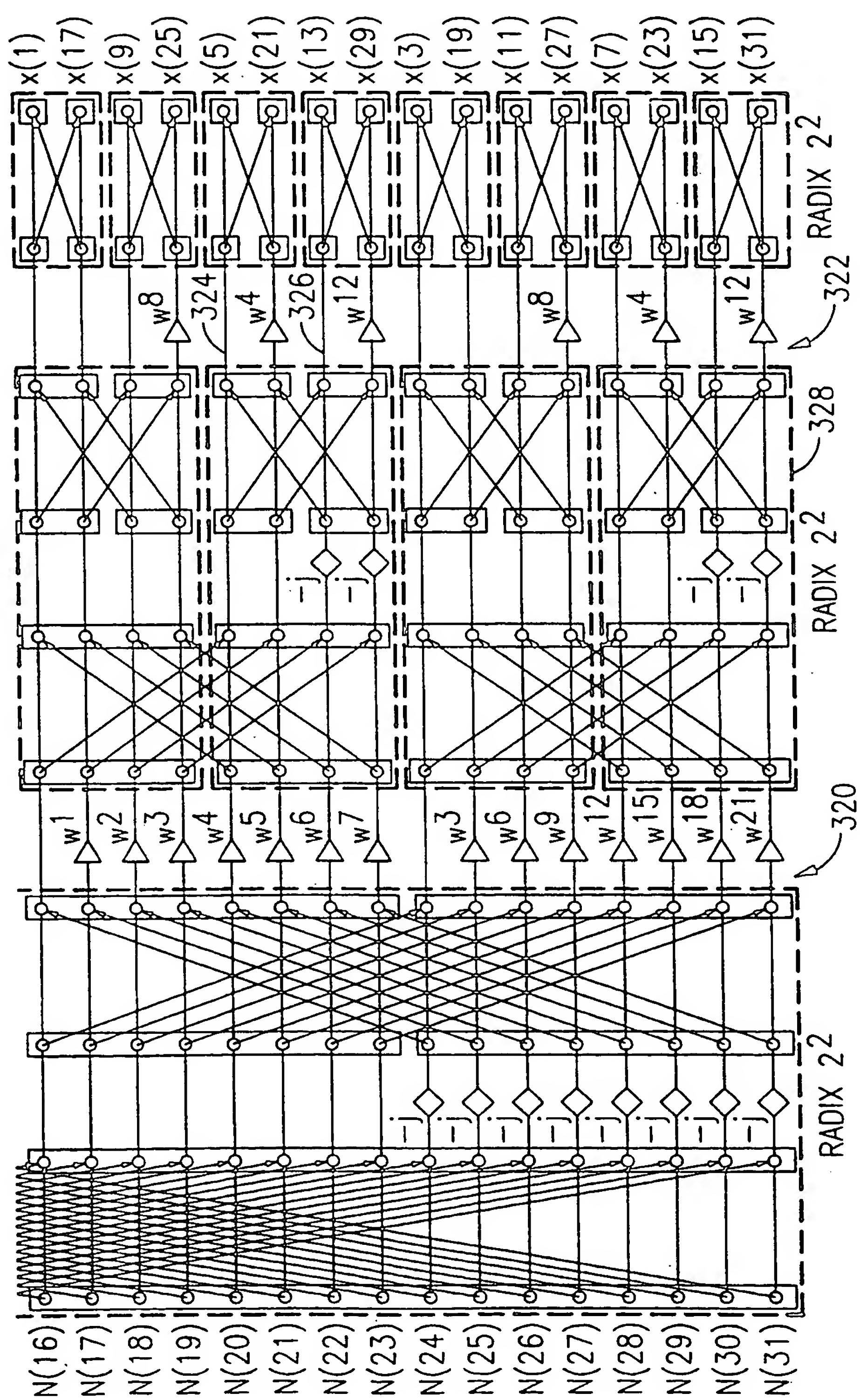


FIG.28A

FIG.28B


x(1)
x(17)
x(9)
x(25)
x(5)
x(21)
x(13)
x(29)
x(3)
x(19)
x(11)
x(27)
x(7)
x(23)
x(15)
x(31)
RADIX 2^2
322
324
326
328
w^8
w^4
w^12
w^8
w^4
w^12
-j
RADIX 2^2
320
w^1
w^2
w^3
w^4
w^5
w^6
w^7
w^3
w^6
w^9
w^12
w^15
w^18
w^21
RADIX 2^2
N(16)
N(17)
N(18)
N(19)
N(20)
N(21)
N(22)
N(23)
N(24)
N(25)
N(26)
N(27)
N(28)
N(29)
N(30)
N(31)

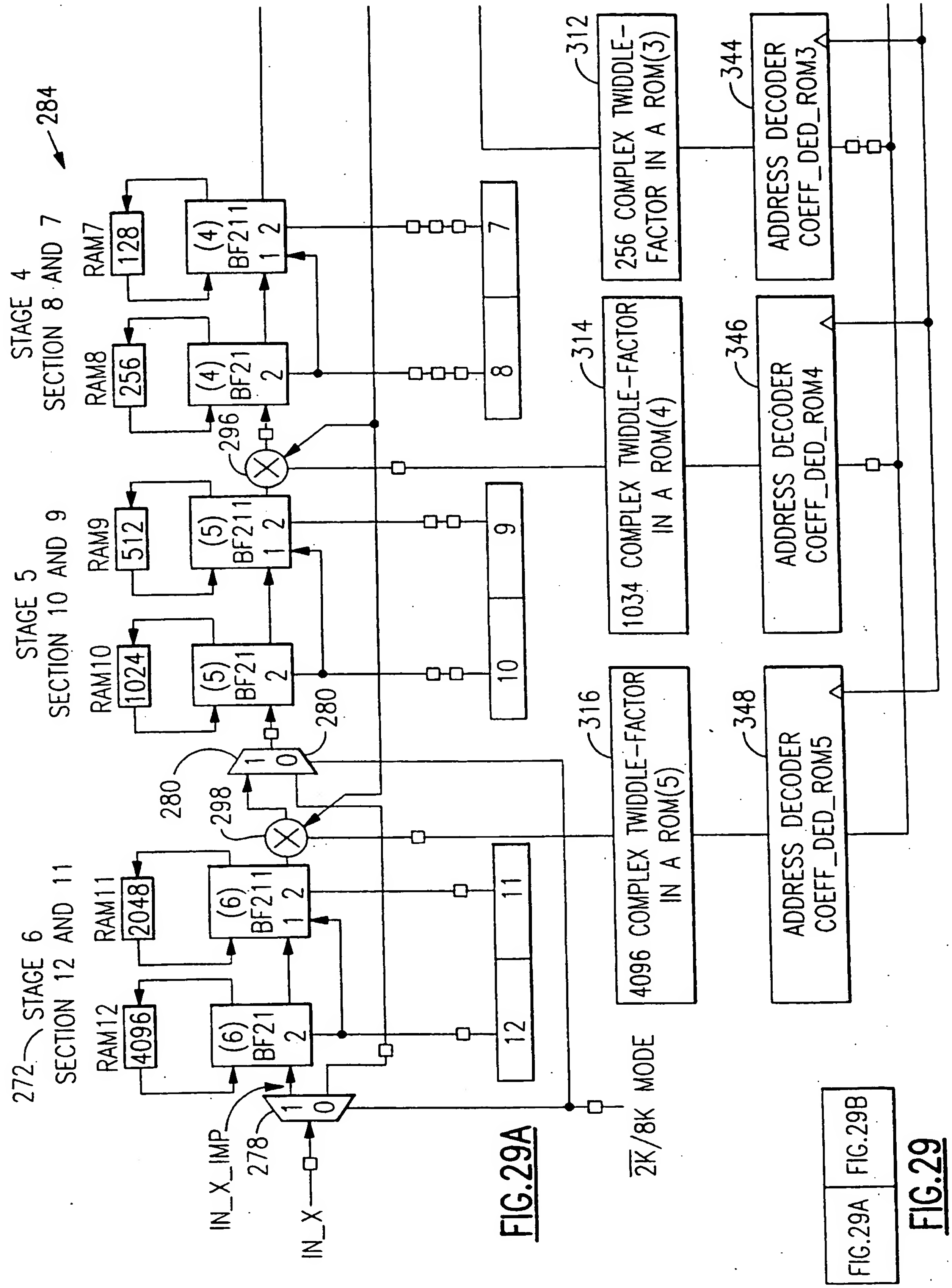


FIG.29A

FIG.29

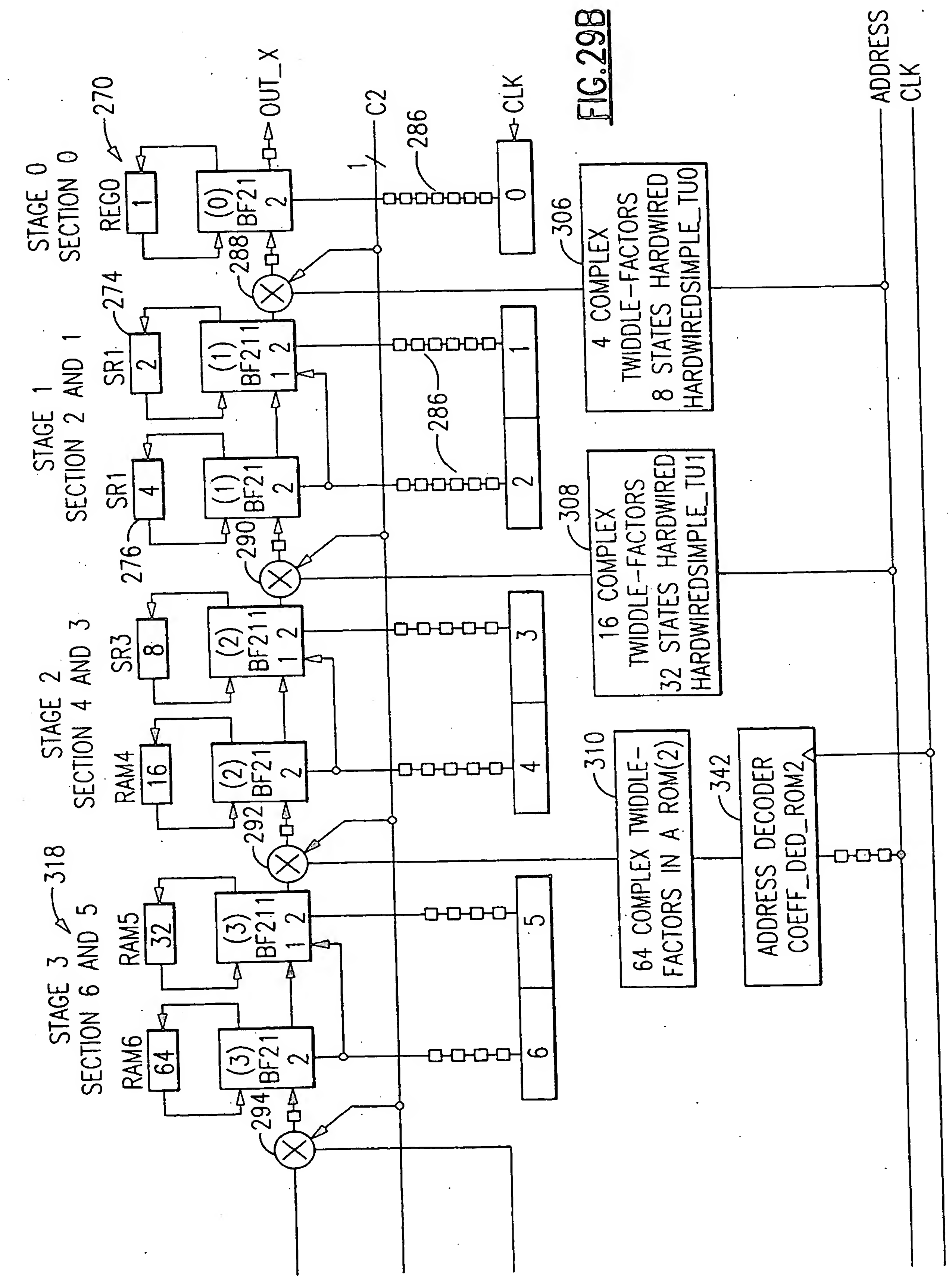

STAGE 0
SECTION 0
REG0
1
270
(0)
BF21
2
OUT_X
C2
1
286
CLK
0
288
STAGE 1
SECTION 2 AND 1
SR1
2
274
(1)
BF211
1 2
SR1
4
276
(1)
BF21
2
290
1
2
306
4 COMPLEX
TWIDDLE-FACTORS
8 STATES HARDWIRED
HARDWIREDSIMPLE_TU0
STAGE 2
SECTION 4 AND 3
SR3
8
(2)
BF211
1 2
RAM4
16
(2)
BF21
2
292
3
4
308
16 COMPLEX
TWIDDLE-FACTORS
32 STATES HARDWIRED
HARDWIREDSIMPLE_TU1
STAGE 3
SECTION 6 AND 5
318
RAM5
32
(3)
BF211
1 2
RAM6
64
(3)
BF21
2
294
5
6
310
64 COMPLEX TWIDDLE-
FACTORS IN A ROM(2)
342
ADDRESS DECODER
COEFF_DED_ROM2
ADDRESS
CLK

FIG.29B

FIG.30

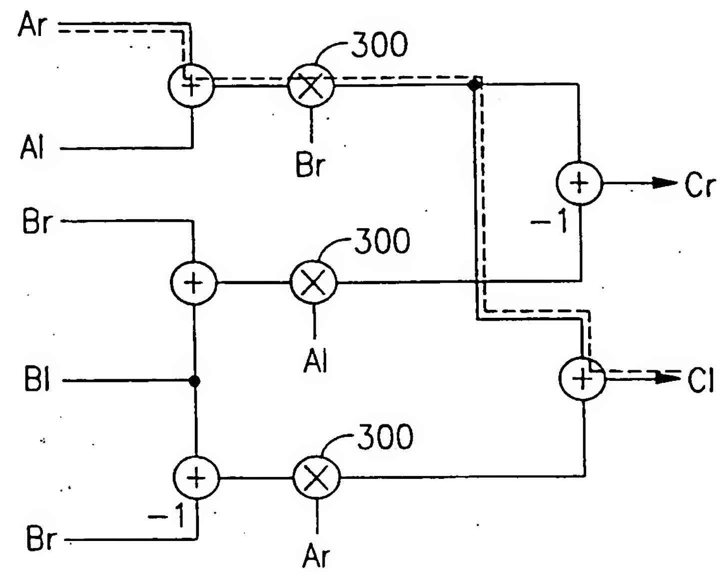
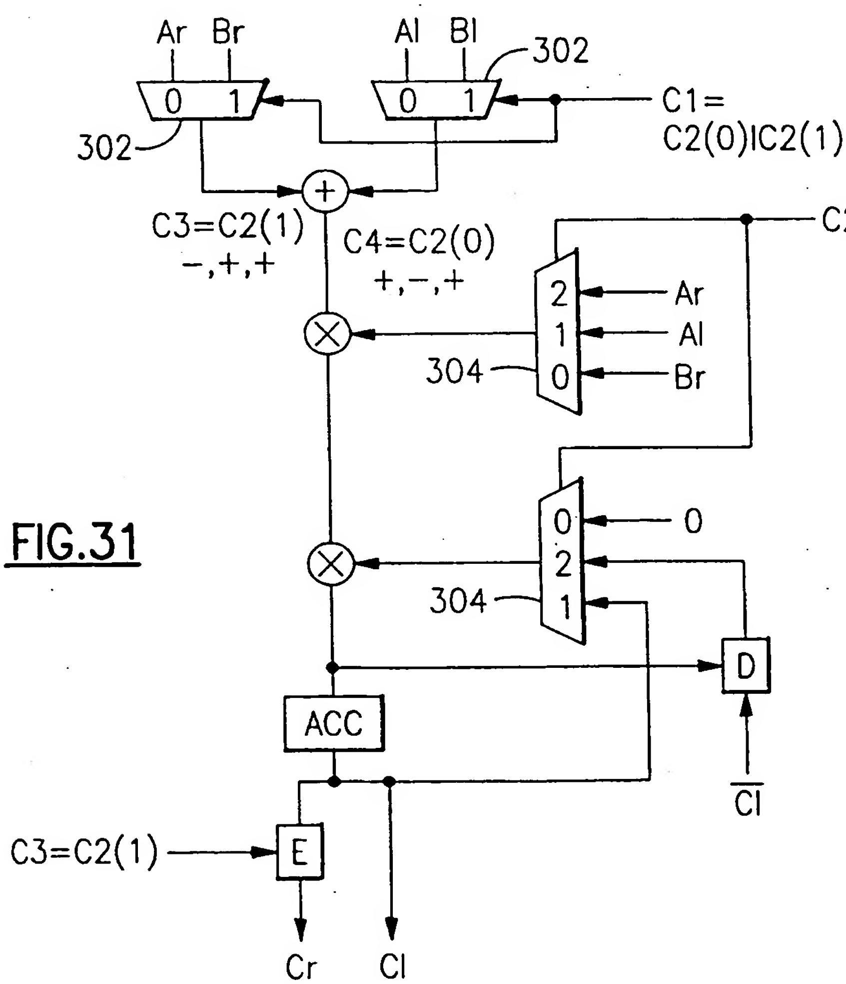

Ar
300
Al
Br
Cr
-1
Br
300
Al
Bl
Cl
300
-1
Br
Ar

FIG.31


Ar
Br
Al
Bl
302
0
1
302
C1=
C2(0)IC2(1)
C3=C2(1)
-,+,+
C4=C2(0)
+,-,+
C2
2
1
0
Ar
Al
Br
304
0
2
1
0
304
D
ACC
Cl
C3=C2(1)
E
Cr
Cl

MULTIPLIER M3

330

w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}
w^{0}	w^{2}	w^{4}	w^{6}	w^{8}	w^{10}	w^{12}	w^{14}	w^{16}	w^{18}	w^{20}	w^{22}	w^{24}	w^{26}	w^{28}	w^{30}
w^{0}	w^{1}	w^{2}	w^{3}	w^{4}	w^{5}	w^{6}	w^{7}	w^{8}	w^{9}	w^{10}	w^{11}	w^{12}	w^{13}	w^{14}	w^{15}
w^{0}	w^{3}	w^{6}	w^{9}	w^{12}	w^{15}	w^{18}	w^{21}	w^{24}	w^{27}	w^{30}	w^{33}	w^{36}	w^{39}	w^{42}	w^{45}

w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}
w^{64}	w^{66}	w^{68}	w^{70}	w^{72}	w^{74}	w^{76}	w^{78}	w^{80}	w^{82}	w^{84}	w^{86}	w^{88}	w^{90}	w^{92}	w^{94}
w^{32}	w^{33}	w^{34}	w^{35}	w^{36}	w^{37}	w^{38}	w^{39}	w^{40}	w^{41}	w^{42}	w^{43}	w^{44}	w^{45}	w^{46}	w^{47}
w^{96}	w^{99}	w^{102}	w^{105}	w^{108}	w^{111}	w^{114}	w^{117}	w^{120}	w^{123}	w^{126}	w^{129}	w^{132}	w^{135}	w^{138}	w^{141}

w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}
w^{128}	w^{130}	w^{132}	w^{134}	w^{136}	w^{138}	w^{140}	w^{142}	w^{144}	w^{146}	w^{148}	w^{150}	w^{152}	w^{154}	w^{156}	w^{158}
w^{64}	w^{65}	w^{66}	w^{67}	w^{68}	w^{69}	w^{70}	w^{71}	w^{72}	w^{73}	w^{74}	w^{75}	w^{76}	w^{77}	w^{78}	w^{79}
w^{192}	w^{195}	w^{198}	w^{201}	w^{204}	w^{207}	w^{210}	w^{213}	w^{216}	w^{219}	w^{222}	w^{225}	w^{228}	w^{231}	w^{234}	w^{237}

w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}	w^{0}
w^{192}	w^{194}	w^{196}	w^{198}	w^{200}	w^{202}	w^{204}	w^{206}	w^{208}	w^{210}	w^{212}	w^{214}	w^{216}	w^{218}	w^{220}	w^{222}
w^{96}	w^{97}	w^{98}	w^{99}	w^{100}	w^{101}	w^{102}	w^{103}	w^{104}	w^{105}	w^{106}	w^{107}	w^{108}	w^{109}	w^{110}	w^{111}
w^{288}	w^{291}	w^{294}	w^{297}	w^{300}	w^{303}	w^{306}	w^{309}	w^{312}	w^{315}	w^{318}	w^{321}	w^{324}	w^{327}	w^{330}	w^{333}

FIG.32A

FIG.32A	FIG.32B

FIG.32

w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0
w^{32}	w^{34}	w^{36}	w^{38}	w^{40}	w^{42}	w^{44}	w^{46}	w^{48}	w^{50}	w^{52}	w^{54}	w^{56}	w^{58}	w^{60}	w^{62}
w^{16}	w^{17}	w^{18}	w^{19}	w^{20}	w^{21}	w^{22}	w^{23}	w^{24}	w^{25}	w^{26}	w^{27}	w^{28}	w^{29}	w^{30}	w^{31}
w^{48}	w^{51}	w^{54}	w^{57}	w^{60}	w^{63}	w^{66}	w^{69}	w^{72}	w^{75}	w^{78}	w^{81}	w^{84}	w^{87}	w^{90}	w^{93}

w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0
w^{96}	w^{98}	w^{100}	w^{102}	w^{104}	w^{106}	w^{108}	w^{110}	w^{112}	w^{114}	w^{116}	w^{118}	w^{120}	w^{122}	w^{124}	w^{126}
w^{48}	w^{49}	w^{50}	w^{51}	w^{52}	w^{53}	w^{54}	w^{55}	w^{56}	w^{57}	w^{58}	w^{59}	w^{60}	w^{61}	w^{62}	w^{63}
w^{144}	w^{147}	w^{150}	w^{153}	w^{156}	w^{159}	w^{162}	w^{165}	w^{168}	w^{171}	w^{174}	w^{177}	w^{180}	w^{183}	w^{186}	w^{189}

w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0
w^{160}	w^{162}	w^{164}	w^{166}	w^{168}	w^{170}	w^{172}	w^{174}	w^{176}	w^{178}	w^{180}	w^{182}	w^{184}	w^{186}	w^{188}	w^{190}
w^{80}	w^{81}	w^{82}	w^{83}	w^{84}	w^{85}	w^{86}	w^{87}	w^{88}	w^{89}	w^{90}	w^{91}	w^{92}	w^{93}	w^{94}	w^{95}
w^{240}	w^{243}	w^{246}	w^{249}	w^{252}	w^{255}	w^{258}	w^{261}	w^{264}	w^{267}	w^{270}	w^{273}	w^{276}	w^{279}	w^{282}	w^{285}

w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0	w^0
w^{224}	w^{226}	w^{228}	w^{230}	w^{232}	w^{234}	w^{236}	w^{238}	w^{240}	w^{242}	w^{244}	w^{246}	w^{248}	w^{250}	w^{252}	w^{254}
w^{112}	w^{113}	w^{114}	w^{115}	w^{116}	w^{117}	w^{118}	w^{119}	w^{120}	w^{121}	w^{122}	w^{123}	w^{124}	w^{125}	w^{126}	w^{127}
w^{336}	w^{339}	w^{342}	w^{345}	w^{348}	w^{351}	w^{354}	w^{357}	w^{360}	w^{363}	w^{366}	w^{369}	w^{372}	w^{375}	w^{378}	w^{381}

FIG.32B

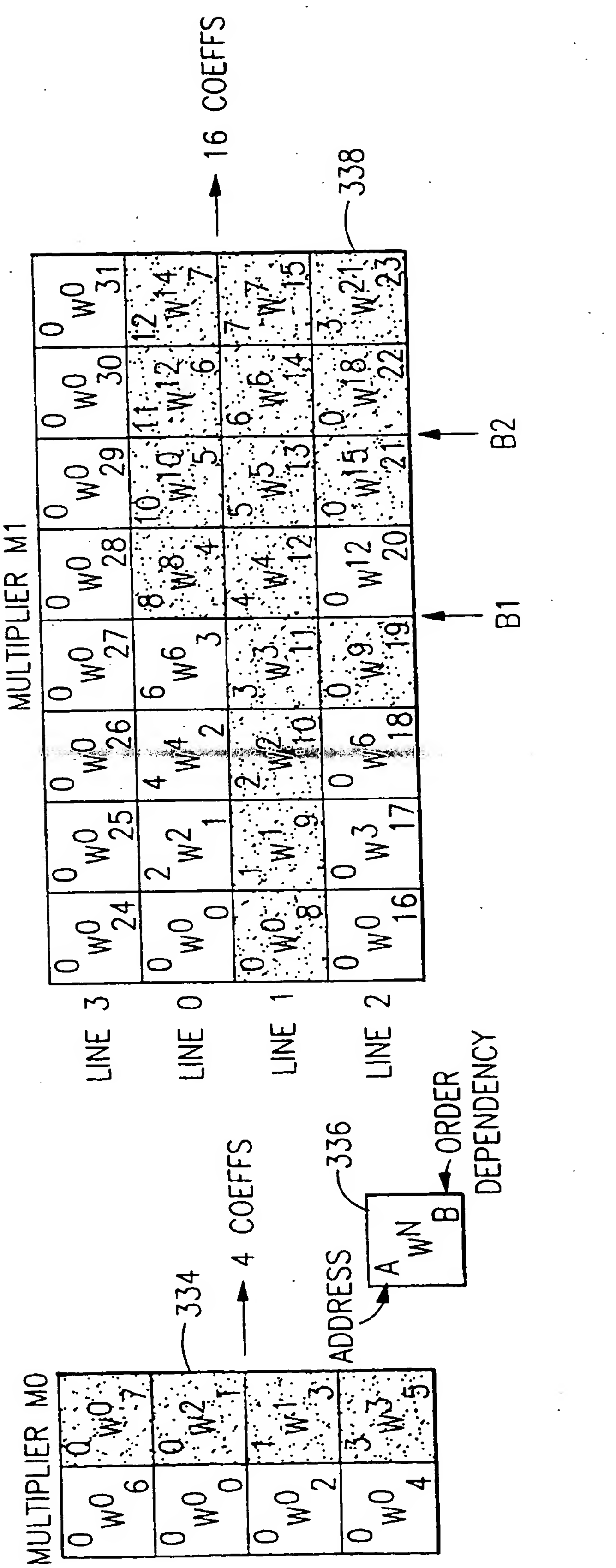

MULTIPLIER M1
16 COEFFS
338
LINE 3
LINE 0
LINE 1
LINE 2
B1
B2
MULTIPLIER M0
334
4 COEFFS
336
ADDRESS
ORDER DEPENDENCY
A
B

FIG.33A

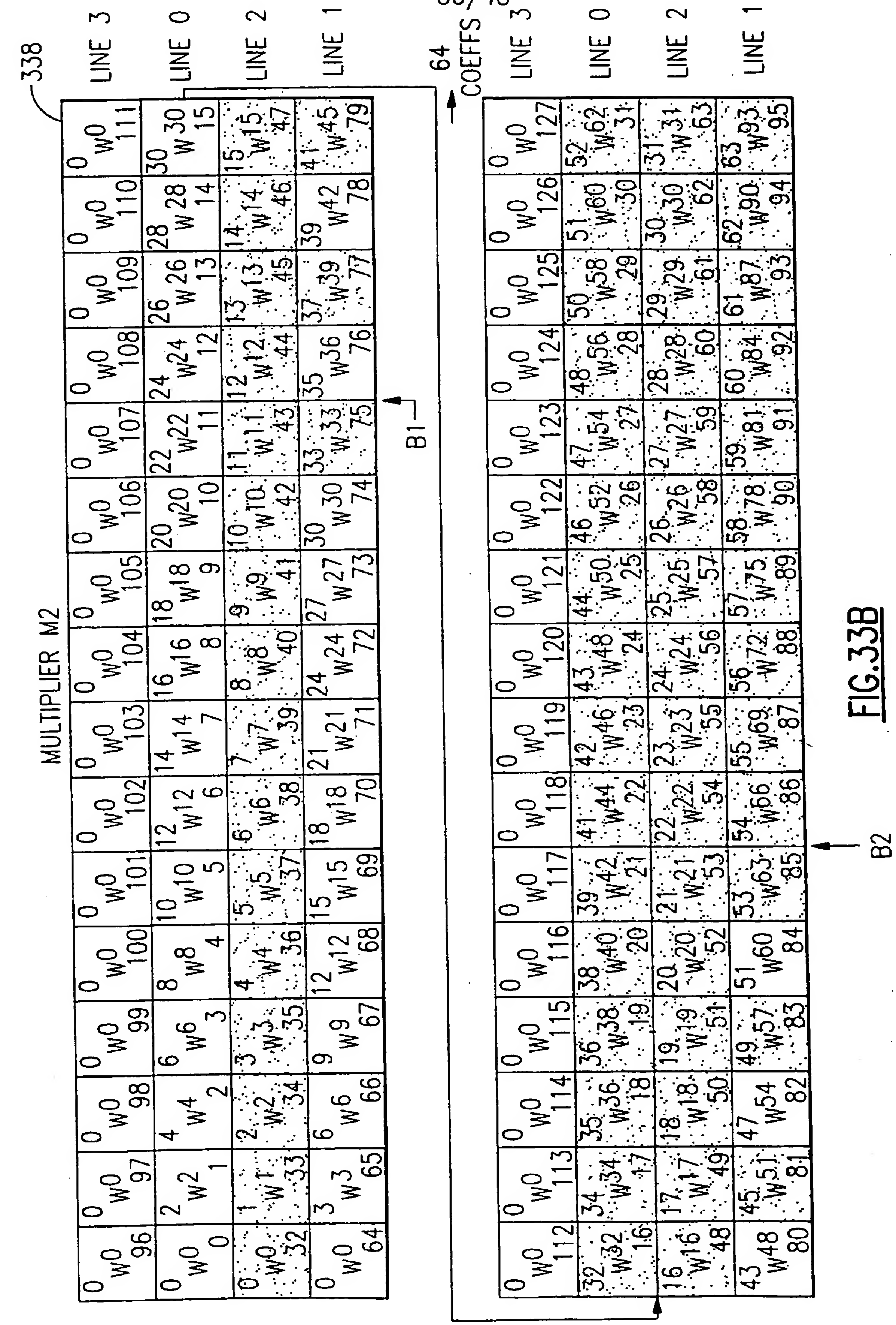


FIG.33B

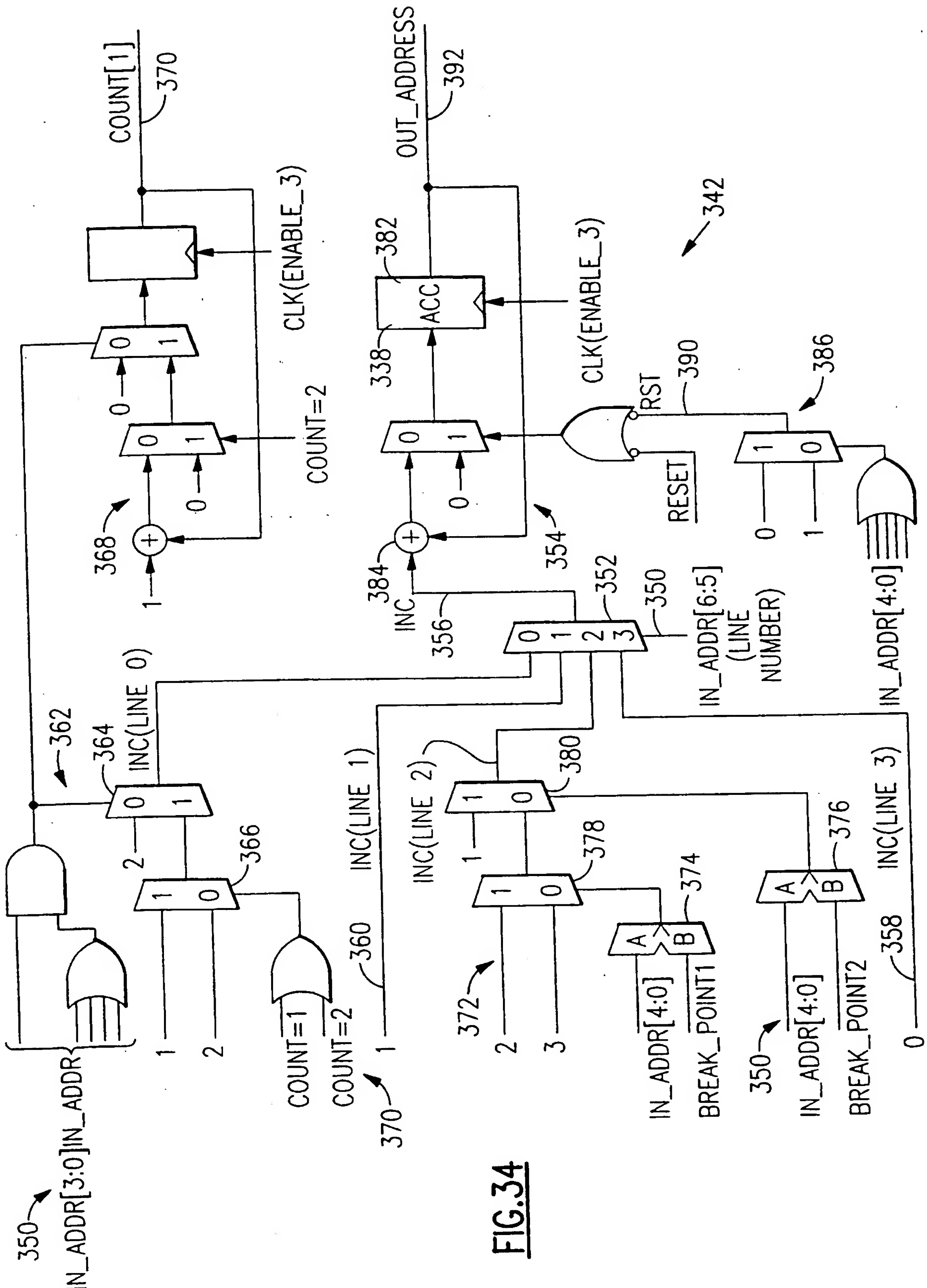

COUNT[1]
370
CLK(ENABLE_3)
COUNT=2
368
OUT_ADDRESS
392
342
382
ACC
338
CLK(ENABLE_3)
RST
390
386
RESET
354
384
INC
356
352
350
IN_ADDR[6:5]
(LINE
NUMBER)
IN_ADDR[4:0]
362
364
INC(LINE 0)
366
INC(LINE 1)
INC(LINE 2)
380
378
374
376
INC(LINE 3)
360
372
358
350
IN_ADDR[3:0]IN_ADDR
COUNT=1
COUNT=2
370
IN_ADDR[4:0]
BREAK_POINT1
350
IN_ADDR[4:0]
BREAK_POINT2

FIG.34

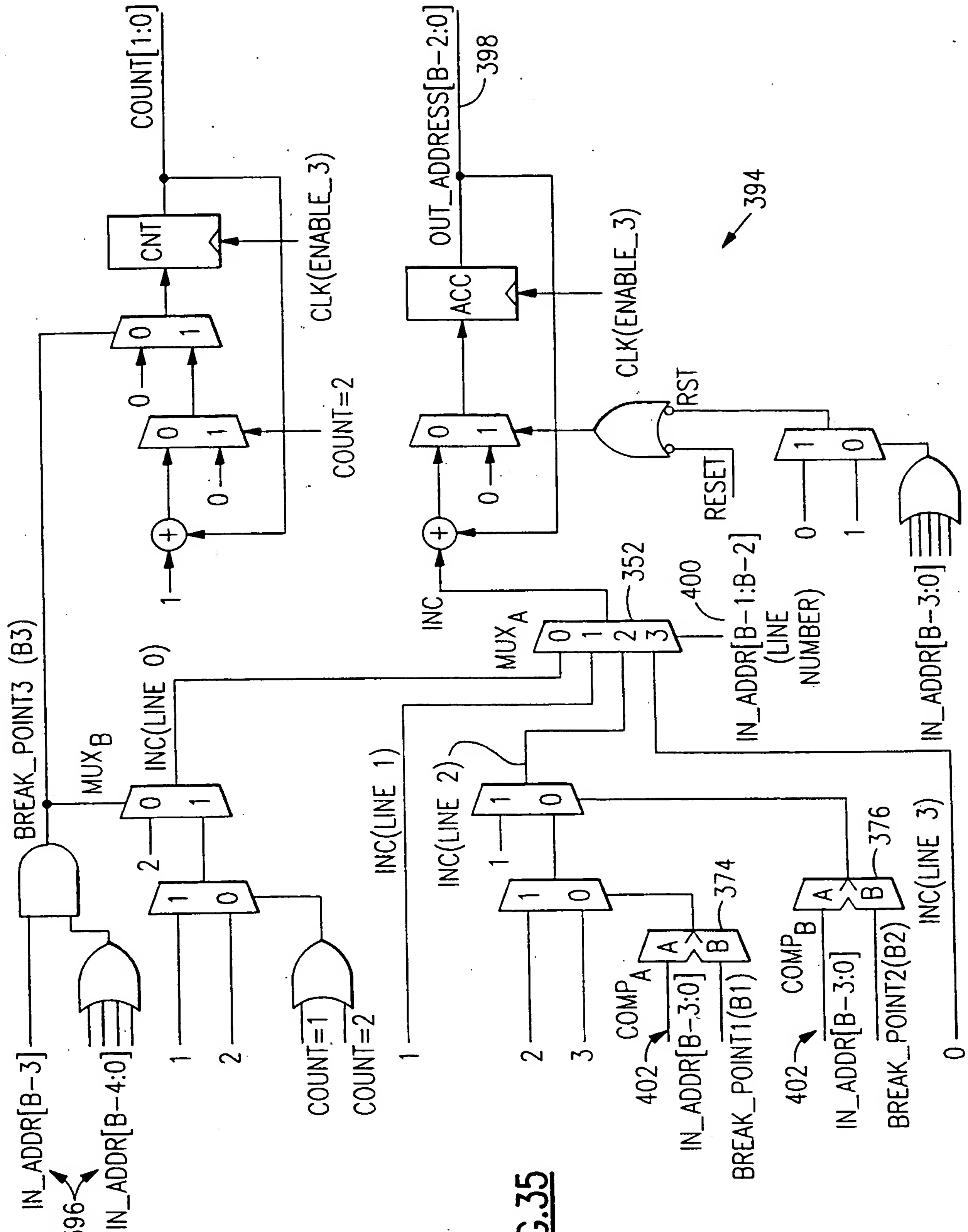

COUNT[1:0]
CNT
CLK(ENABLE_3)
COUNT=2
OUT_ADDRESS[B-2:0]
398
ACC
394
RST
RESET
INC
MUX_A
352
400
IN_ADDR[B-1:B-2] (LINE NUMBER)
IN_ADDR[B-3:0]
BREAK_POINT3 (B3)
MUX_B
INC(LINE 0)
INC(LINE 1)
INC(LINE 2)
INC(LINE 3)
374
376
COMP_A
COMP_B
402
IN_ADDR[B-3:0]
BREAK_POINT1(B1)
BREAK_POINT2(B2)
IN_ADDR[B-3]
IN_ADDR[B-4:0]
396
COUNT=1
COUNT=2

FIG.35

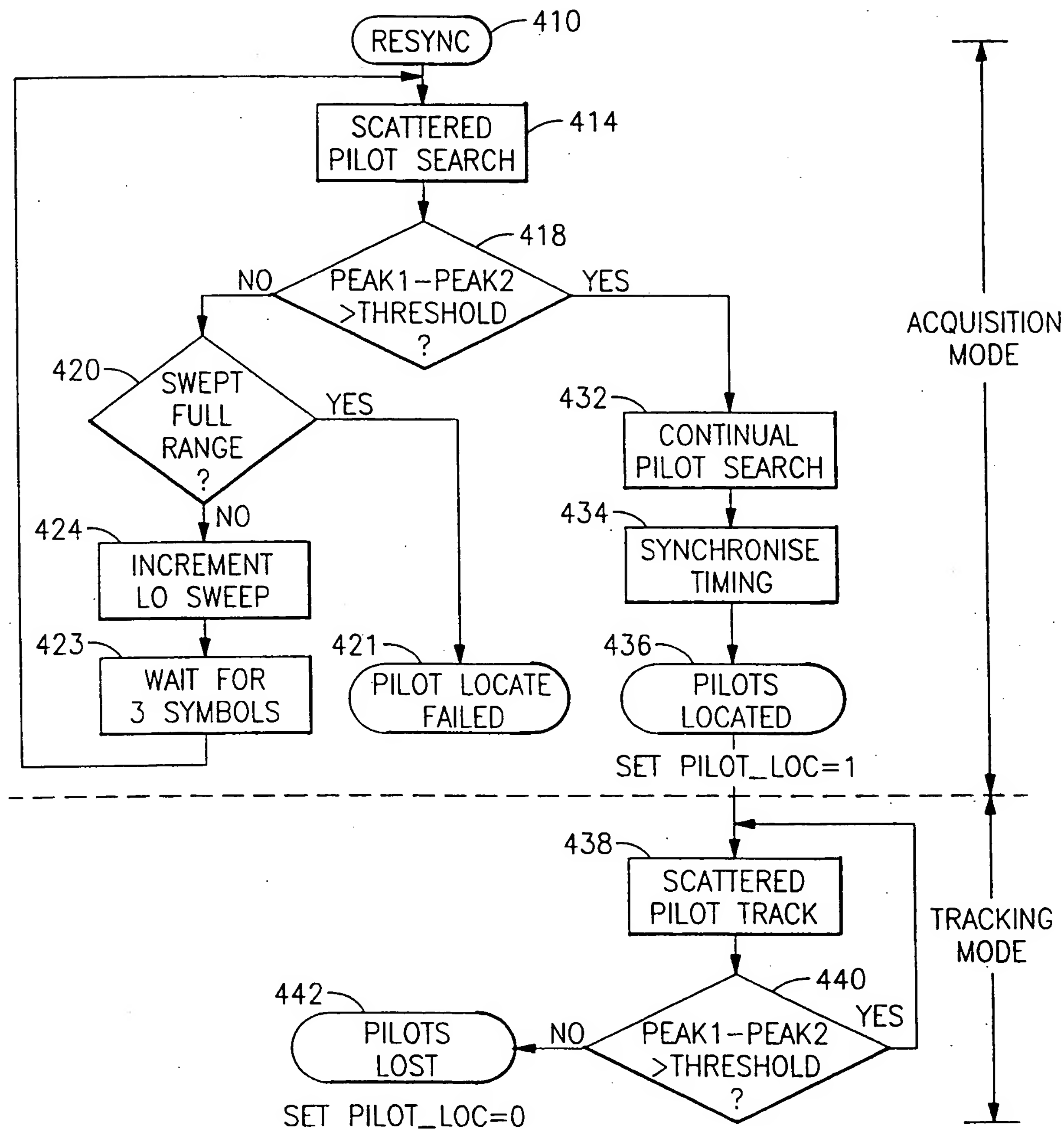

RESYNC
410
SCATTERED PILOT SEARCH
414
418
PEAK1-PEAK2 >THRESHOLD ?
NO
YES
420
SWEPT FULL RANGE ?
YES
NO
424
INCREMENT LO SWEEP
423
WAIT FOR 3 SYMBOLS
421
PILOT LOCATE FAILED
432
CONTINUAL PILOT SEARCH
434
SYNCHRONISE TIMING
436
PILOTS LOCATED
SET PILOT_LOC=1
ACQUISITION MODE
438
SCATTERED PILOT TRACK
440
PEAK1-PEAK2 >THRESHOLD ?
YES
NO
442
PILOTS LOST
SET PILOT_LOC=0
TRACKING MODE

FIG.36

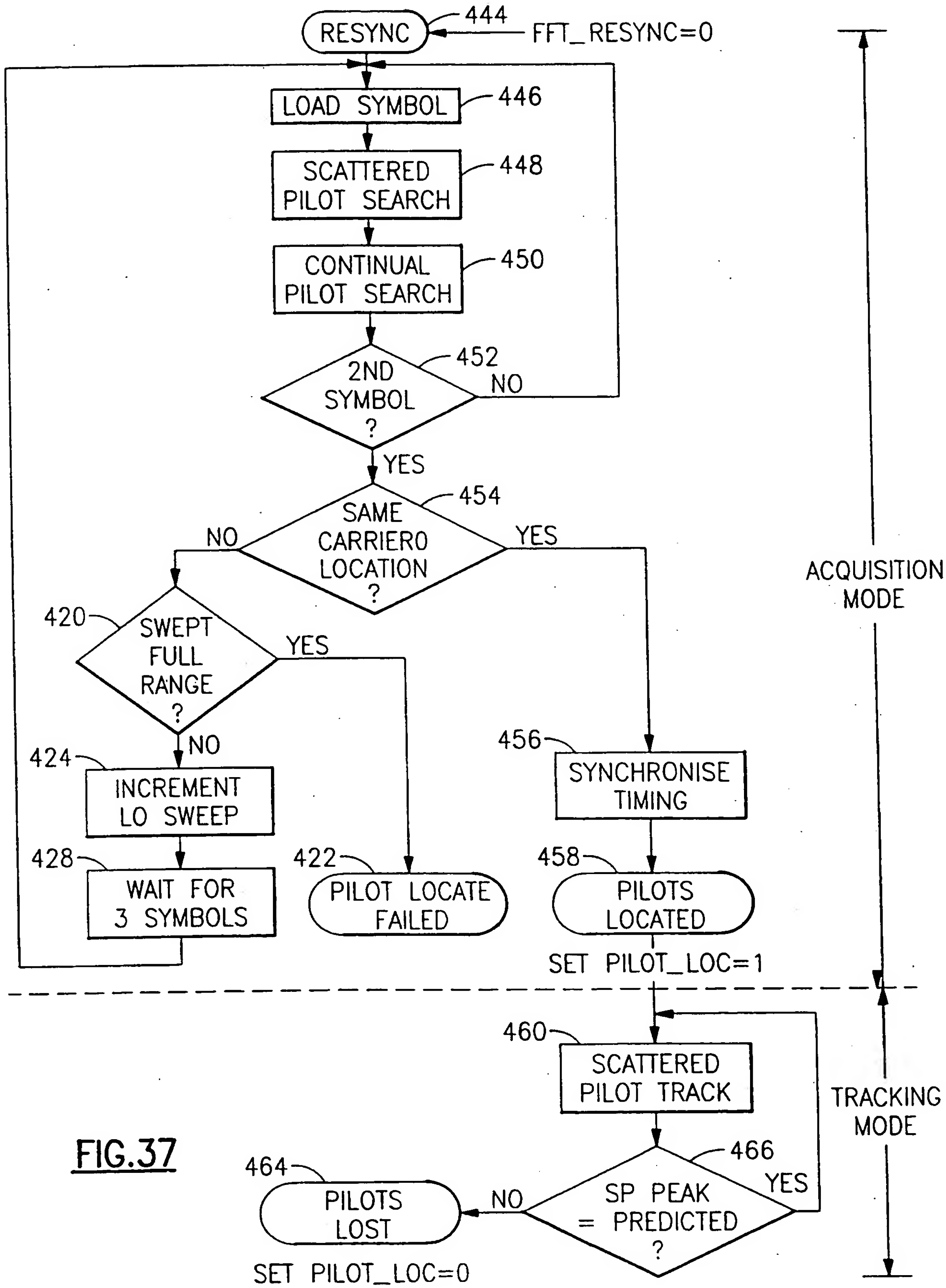

RESYNC
444
FFT_RESYNC=0
LOAD SYMBOL
446
SCATTERED PILOT SEARCH
448
CONTINUAL PILOT SEARCH
450
2ND SYMBOL ?
452
NO
YES
SAME CARRIER0 LOCATION ?
454
NO
YES
420
SWEPT FULL RANGE ?
YES
NO
424
INCREMENT LO SWEEP
428
WAIT FOR 3 SYMBOLS
422
PILOT LOCATE FAILED
456
SYNCHRONISE TIMING
458
PILOTS LOCATED
SET PILOT_LOC=1
ACQUISITION MODE
460
SCATTERED PILOT TRACK
TRACKING MODE
464
PILOTS LOST
NO
SP PEAK = PREDICTED ?
466
YES
SET PILOT_LOC=0

FIG.37

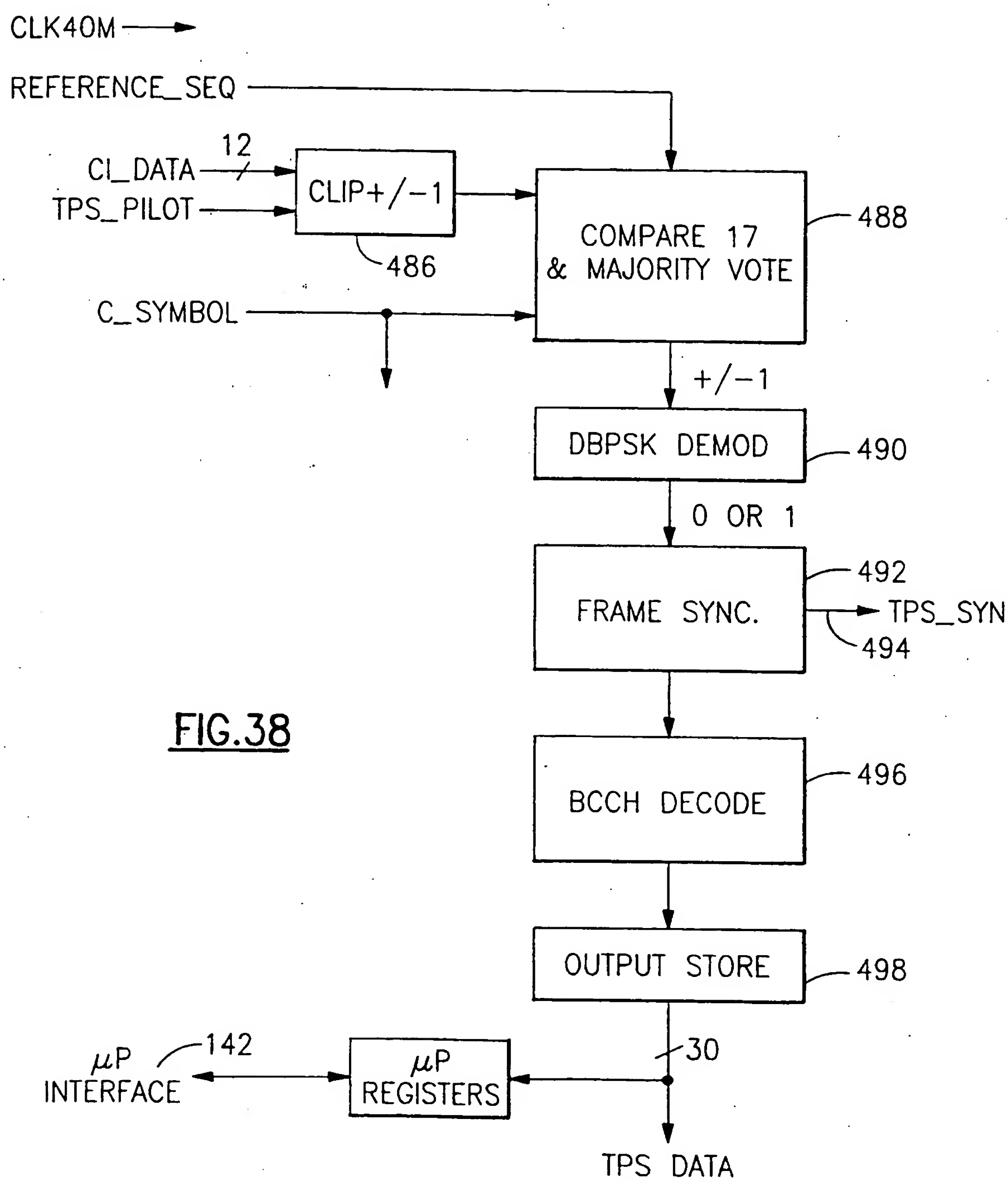

CLK40M
REFERENCE_SEQ
CI_DATA
12
TPS_PILOT
CLIP+/-1
486
COMPARE 17 & MAJORITY VOTE
488
C_SYMBOL
+/-1
DBPSK DEMOD
490
0 OR 1
FRAME SYNC.
492
TPS_SYN
494
BCCH DECODE
496
OUTPUT STORE
498
30
μP INTERFACE
142
μP REGISTERS
TPS DATA

FIG.38

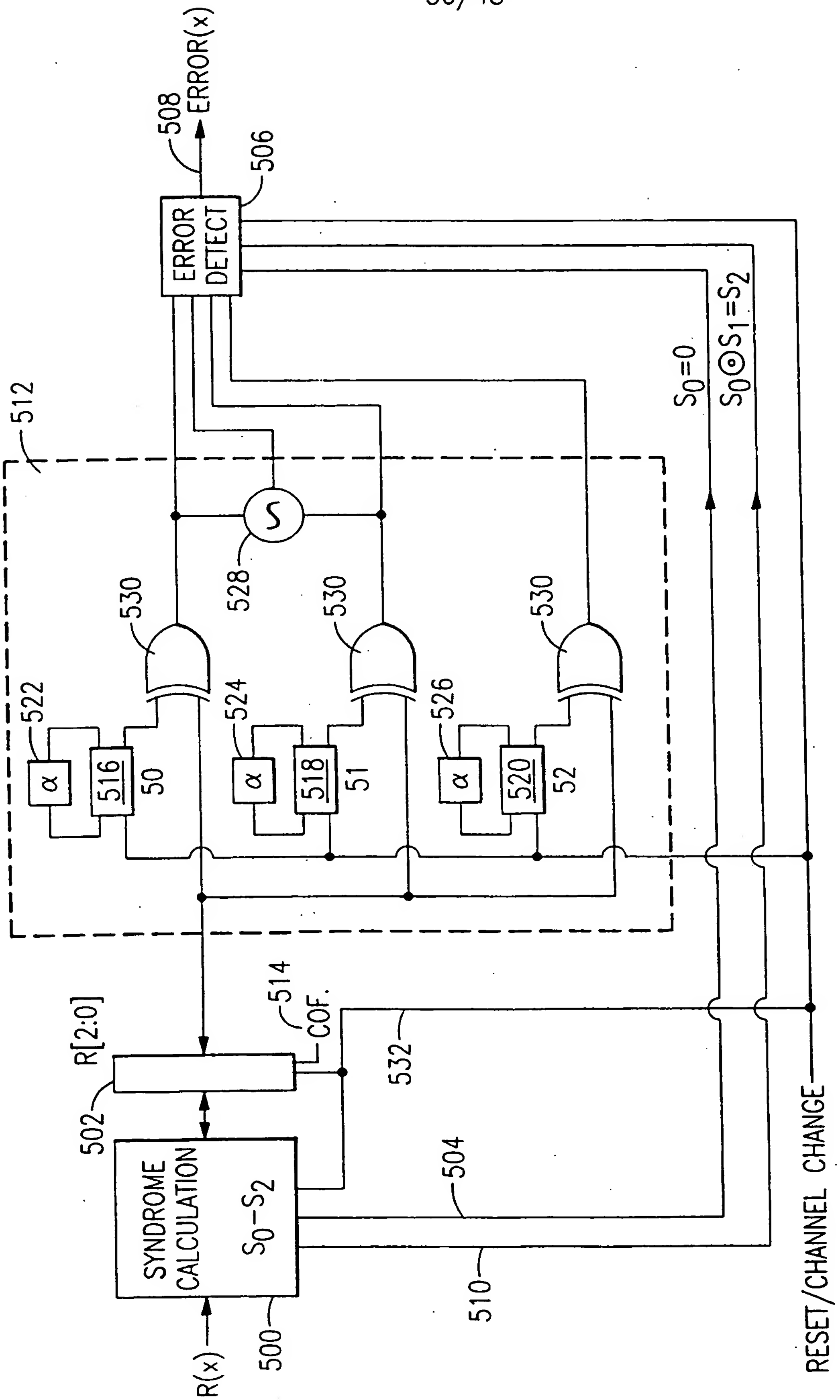

ERROR(x)
508
506
ERROR DETECT
512
S0=0
S0⊙S1=S2
530
528
522
α
516
50
524
518
51
526
520
52
R[2:0]
502
514
COF.
532
SYNDROME CALCULATION
S0-S2
504
510
R(x)
500
RESET/CHANNEL CHANGE

FIG.39

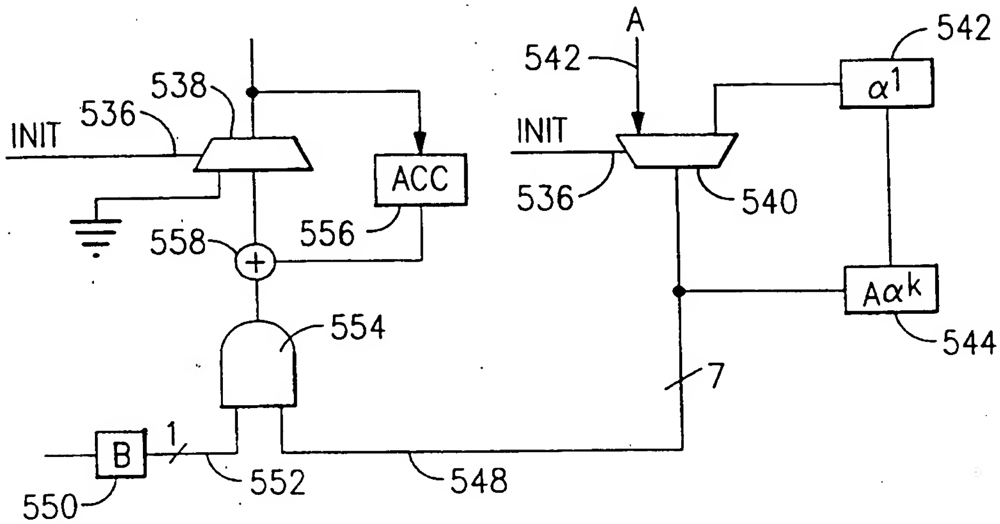

INIT
536
538
ACC
556
558
554
B
1
550
552
548
7
A
542
INIT
536
540
α^1
542
$A\alpha^k$
544

FIG.40

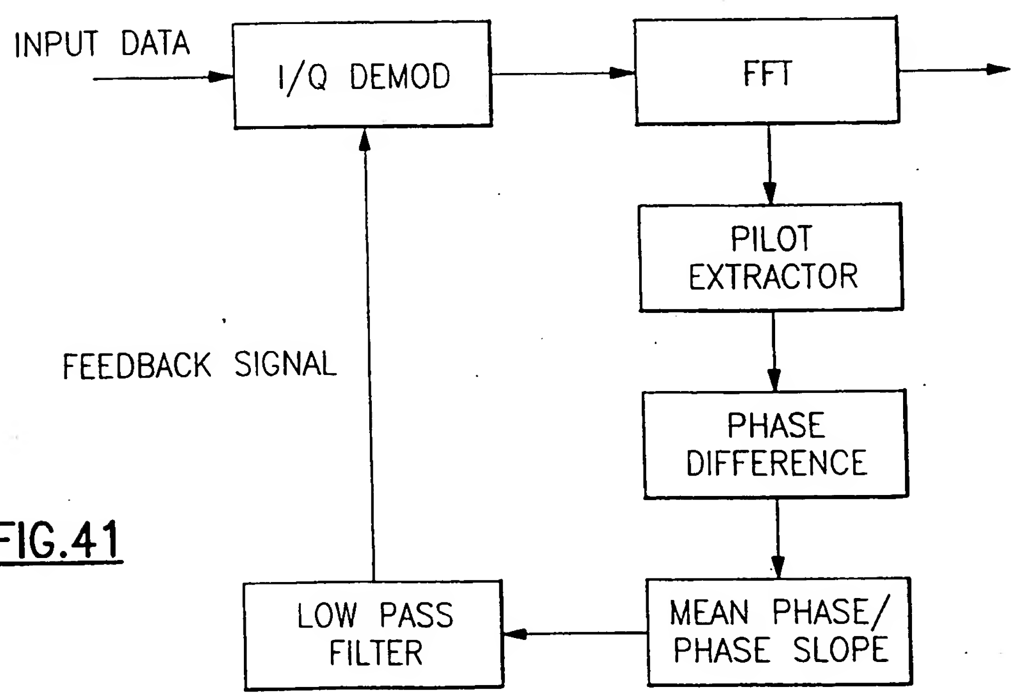
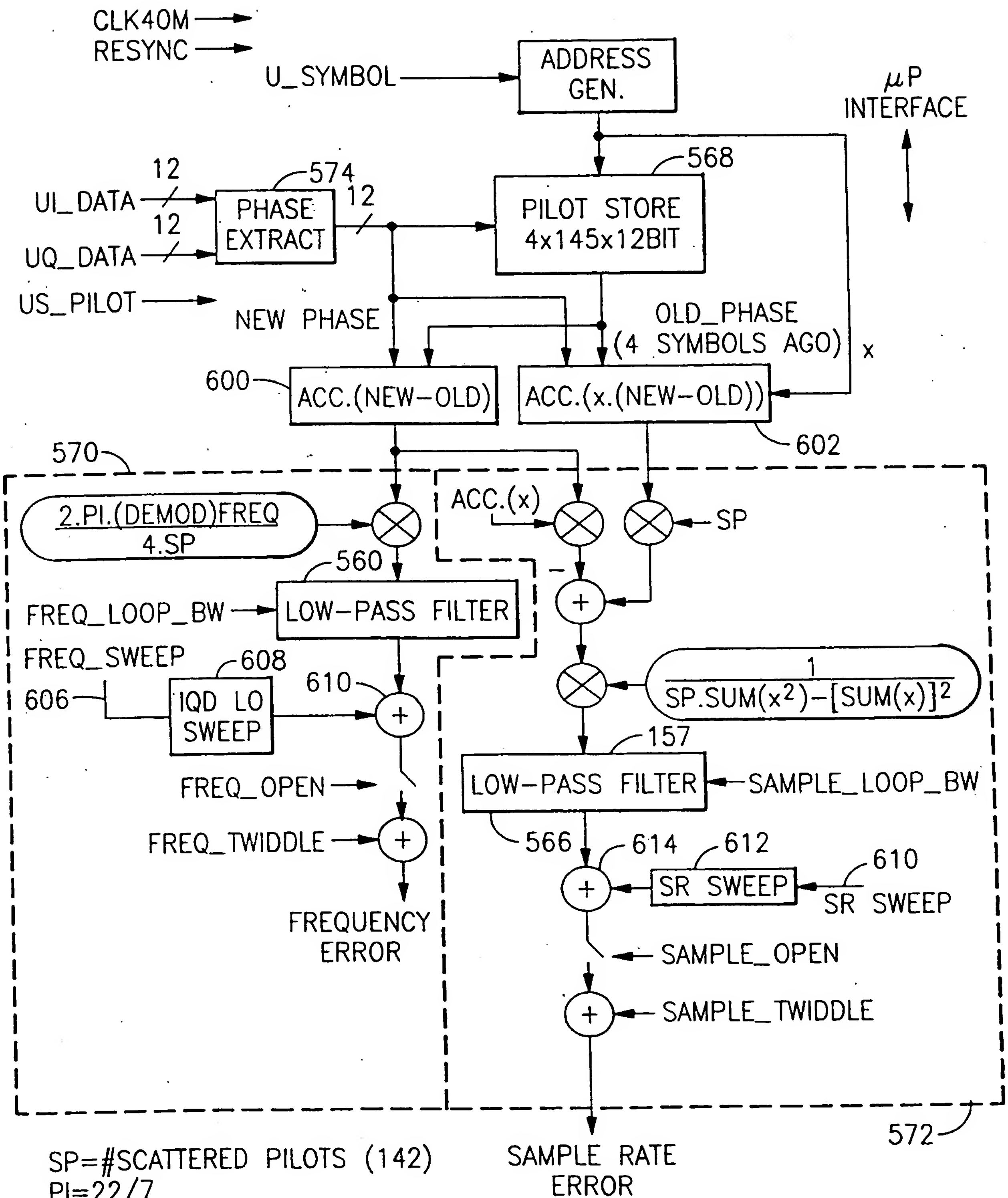

INPUT DATA
I/Q DEMOD
FFT
PILOT EXTRACTOR
PHASE DIFFERENCE
MEAN PHASE/ PHASE SLOPE
LOW PASS FILTER
FEEDBACK SIGNAL

FIG.41



SP=#SCATTERED PILOTS (142)
PI=22/7
DEMOD=32/7MHz
ACC.=ACCUMULATION OF SP SAMPLES

FIG.42

39/48

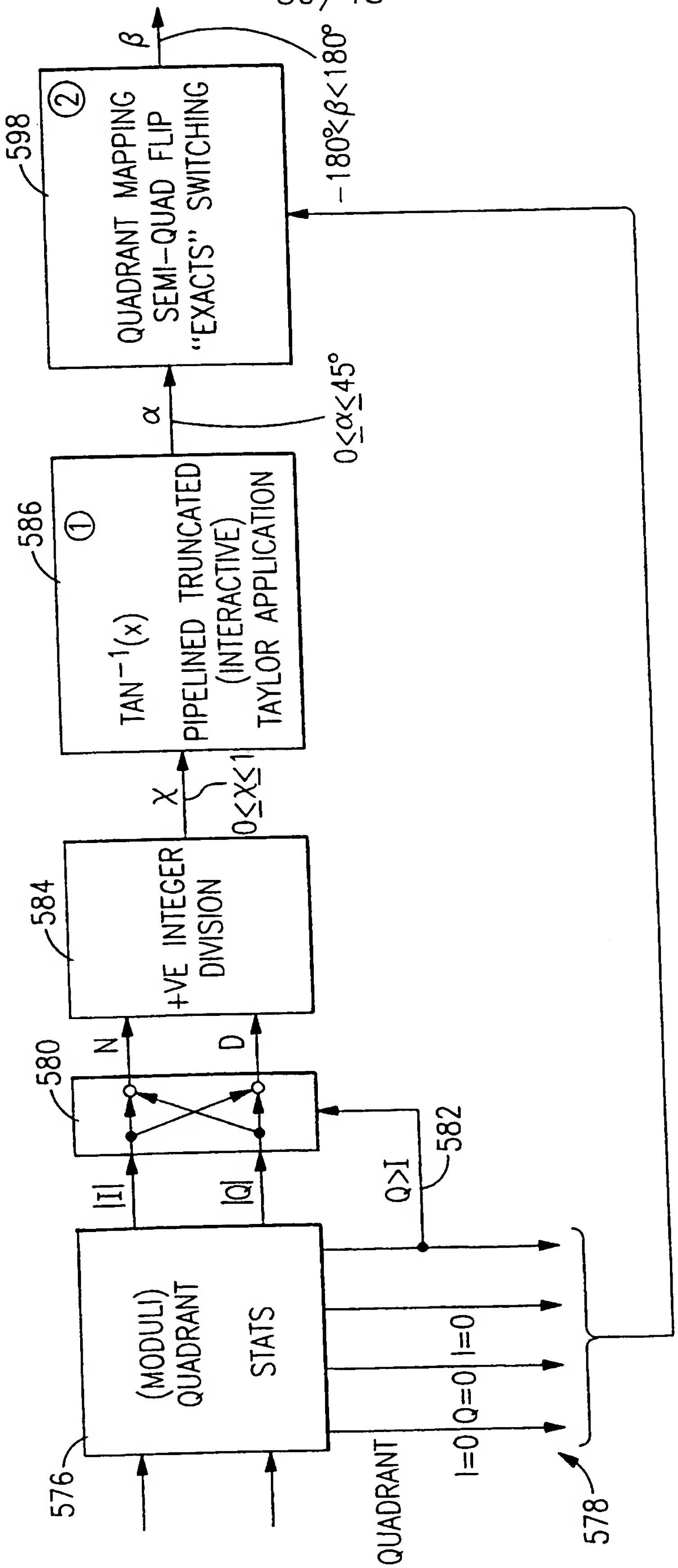

576
(MODULI) QUADRANT STATS
|I|
|Q|
QUADRANT
I=0
Q=0
I=0
578
580
N
D
Q>I
582
584
+VE INTEGER DIVISION
X
0≤X≤1
586
①
TAN^{-1}(x)
PIPELINED TRUNCATED (INTERACTIVE) TAYLOR APPLICATION
α
0≤α≤45°
598
②
QUADRANT MAPPING
SEMI-QUAD FLIP
"EXACTS" SWITCHING
β
−180°≤β<180°

FIG.43

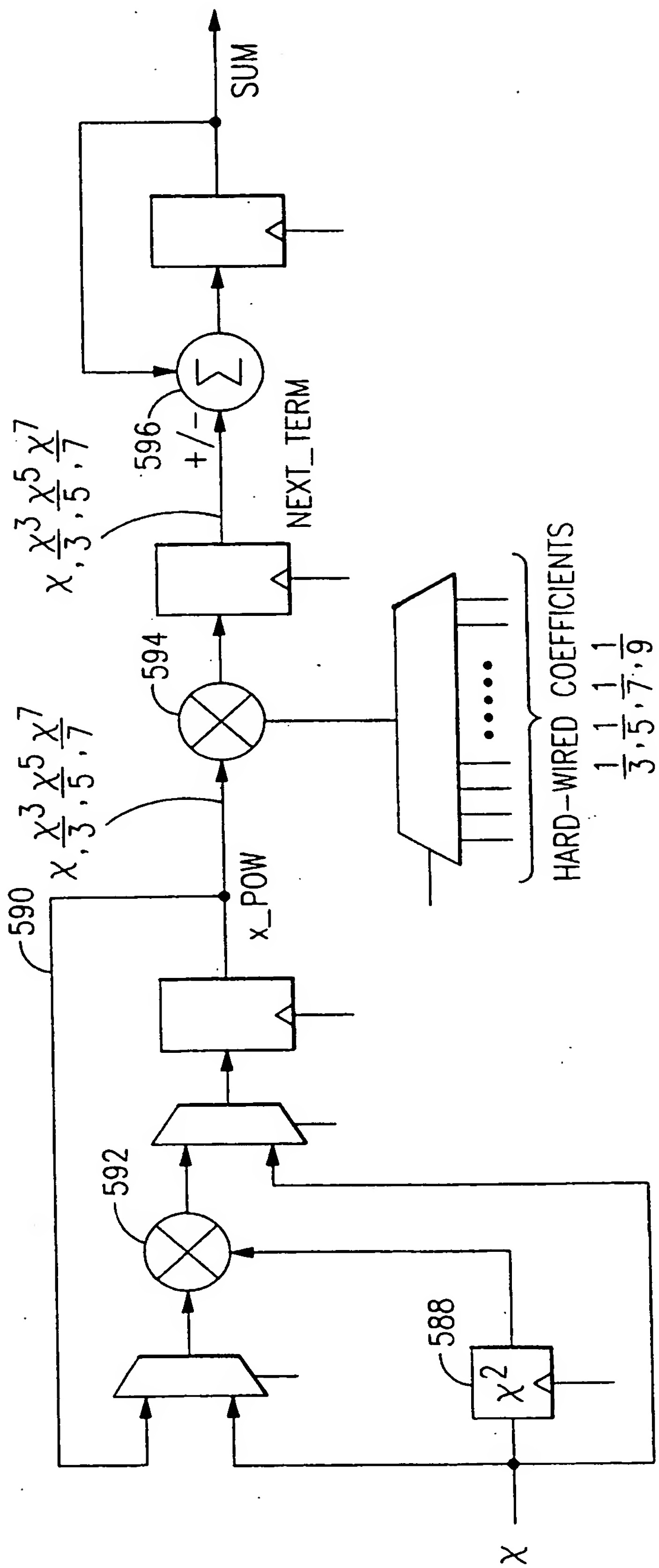

SUM
596
+/-
NEXT_TERM
594
HARD-WIRED COEFFICIENTS
$\frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}$
$\chi, \frac{\chi^3}{3}, \frac{\chi^5}{5}, \frac{\chi^7}{7}$
$\chi, \frac{\chi^3}{3}, \frac{\chi^5}{5}, \frac{\chi^7}{7}$
590
x_POW
592
588
χ^2
χ

FIG.44

FIG.45

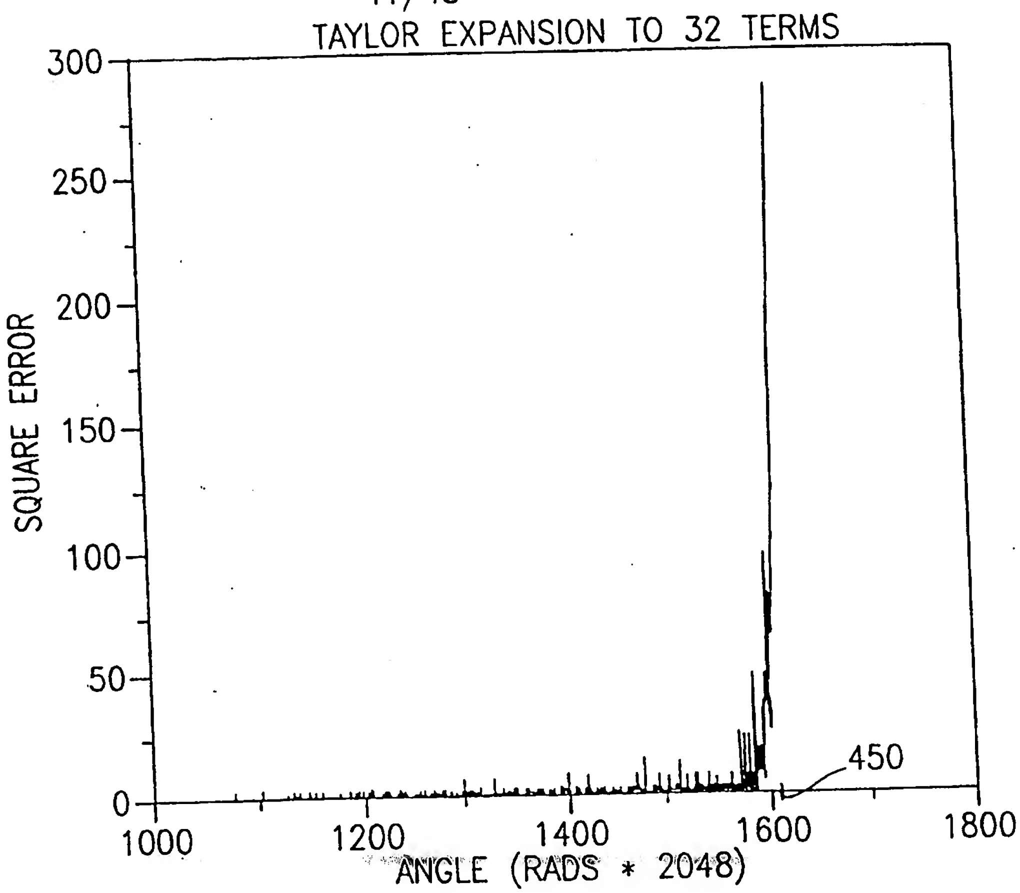

TAYLOR EXPANSION TO 32 TERMS
SQUARE ERROR
300
250
200
150
100
50
0
1000
1200
1400
1600
1800
ANGLE (RADS * 2048)
450

FIG.46

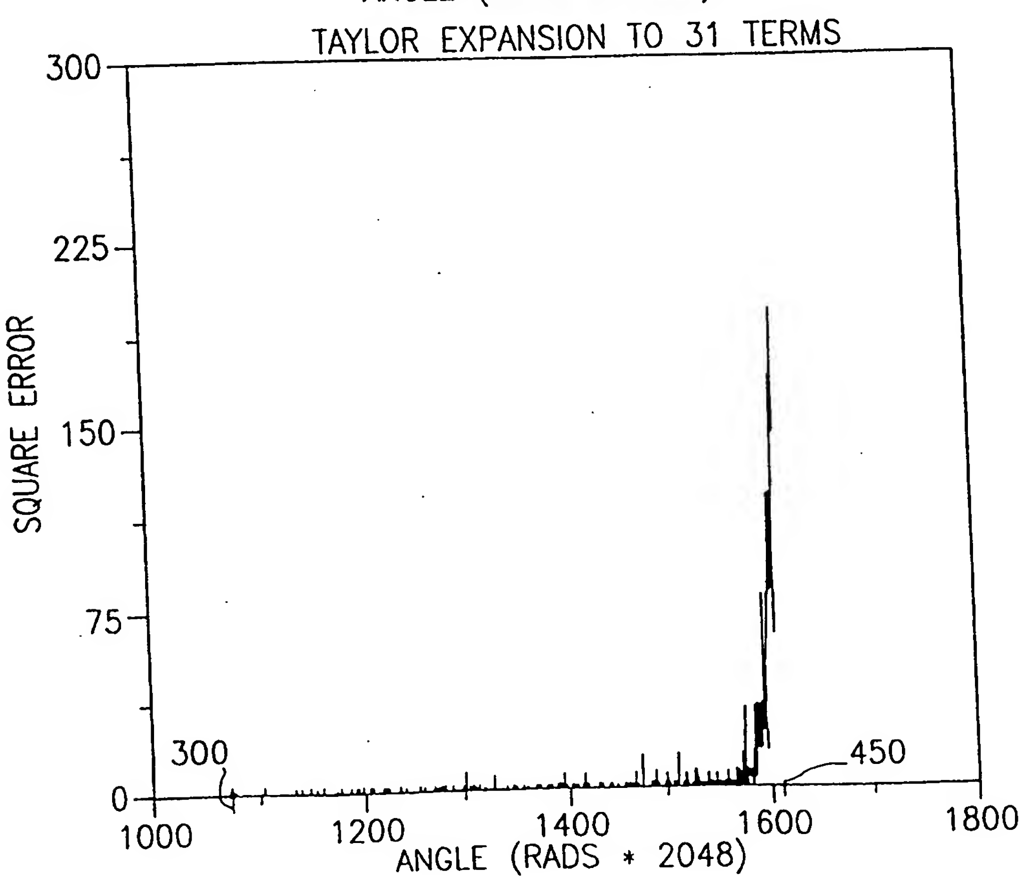
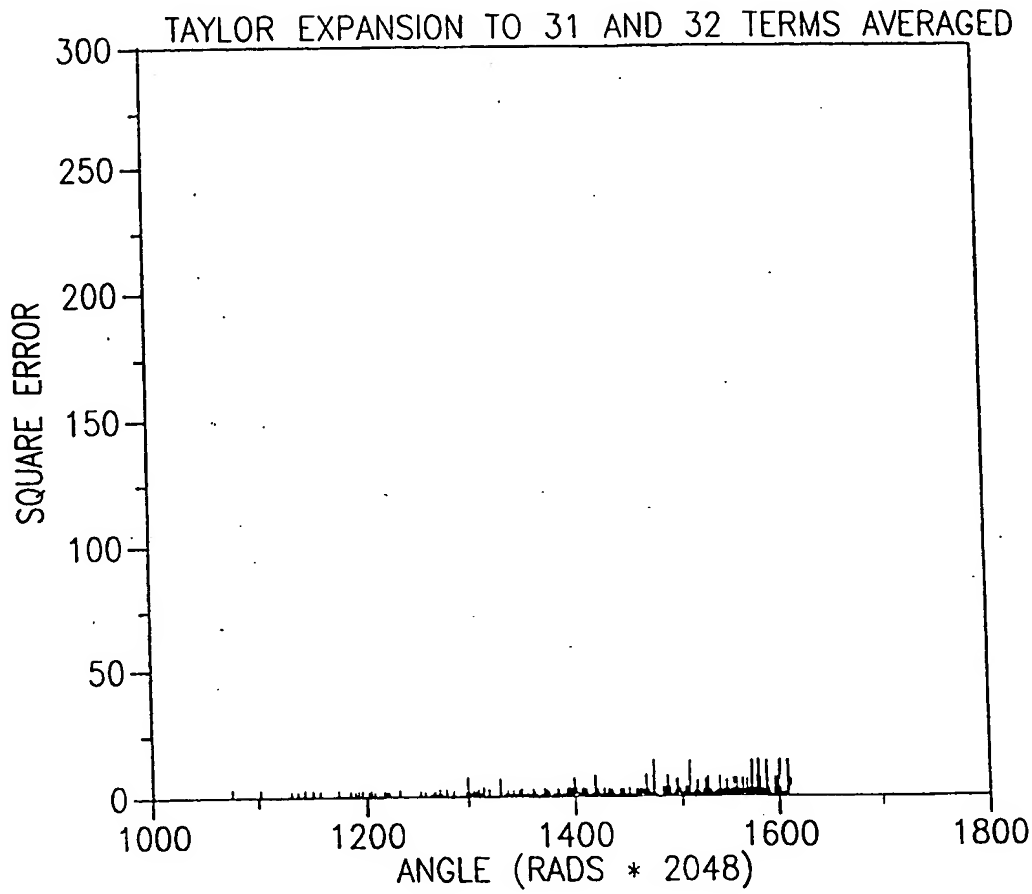

TAYLOR EXPANSION TO 31 TERMS
SQUARE ERROR
300
225
150
75
0
1000
1200
1400
1600
1800
ANGLE (RADS * 2048)
300
450

FIG.47


TAYLOR EXPANSION TO 31 AND 32 TERMS AVERAGED
SQUARE ERROR
300
250
200
150
100
50
0
1000
1200
1400
1600
1800
ANGLE (RADS * 2048)

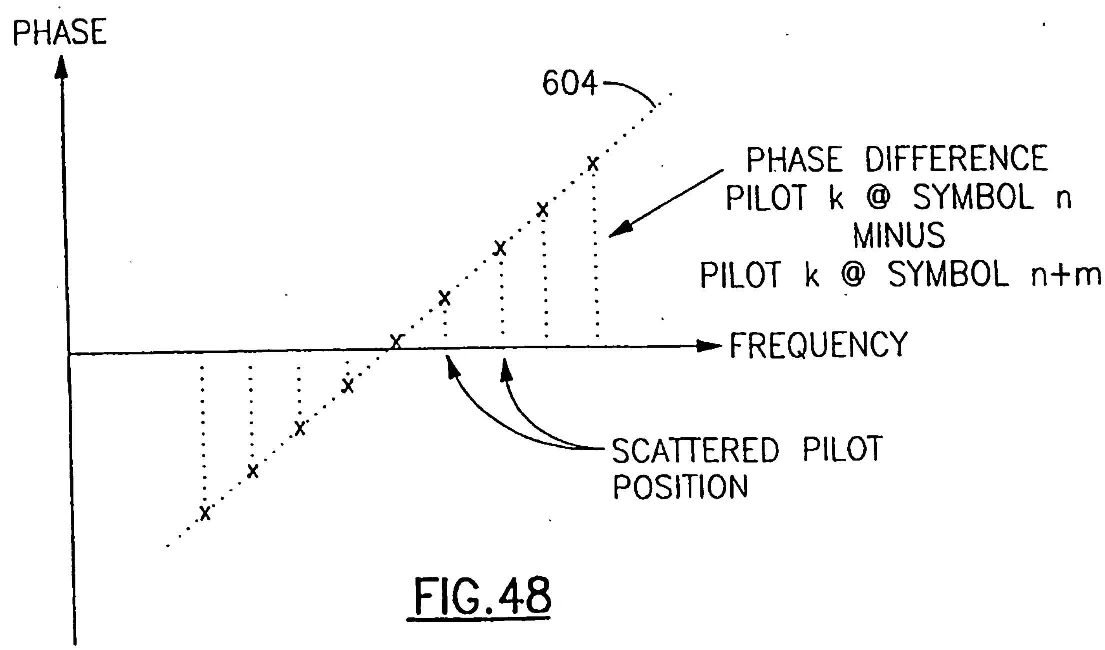
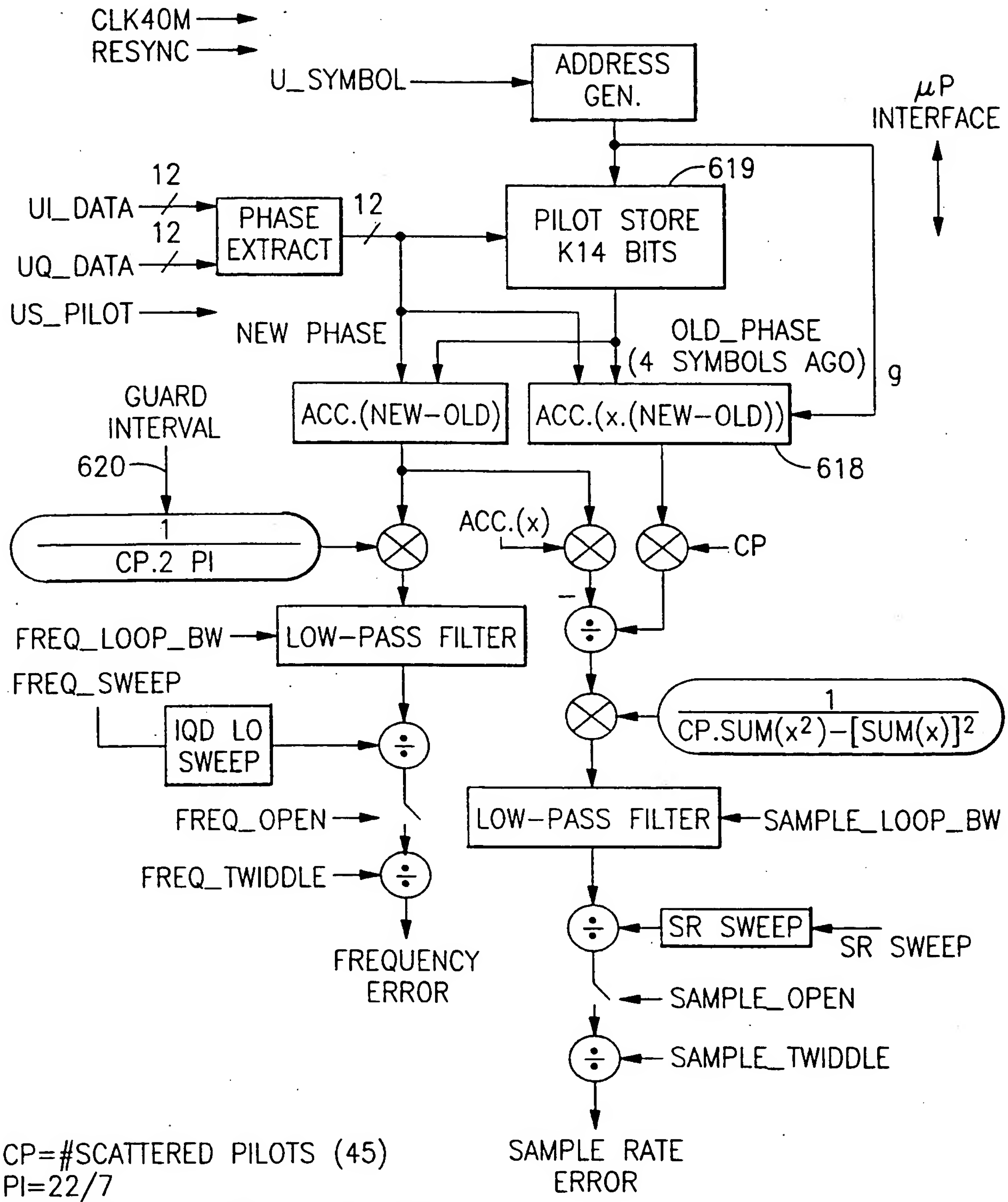

PHASE
604
PHASE DIFFERENCE
PILOT k @ SYMBOL n
MINUS
PILOT k @ SYMBOL n+m
FREQUENCY
SCATTERED PILOT
POSITION

FIG.48



CP=#SCATTERED PILOTS (45)
PI=22/7
ACC.=ACCUMULATION OF SP SAMPLES
Tt=SYMBOL PERIOD (DEPENDS ON GUARD)
m=4(SYMBOLS BEFORE SP "PHASE" REPEATS)

FIG.49

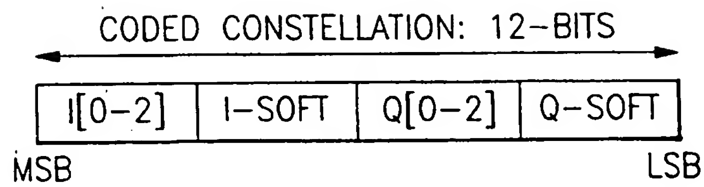

CODED CONSTELLATION: 12-BITS
I[0-2]
I-SOFT
Q[0-2]
Q-SOFT
MSB
LSB

FIG.50

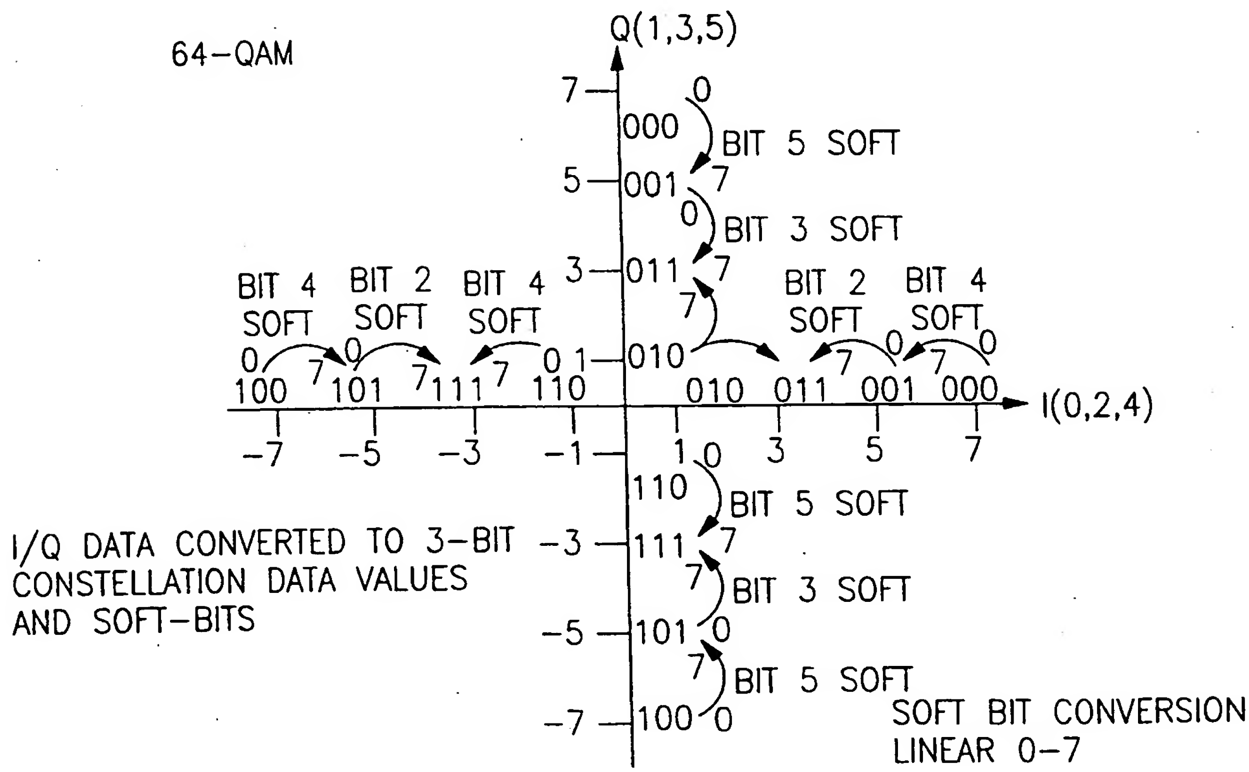
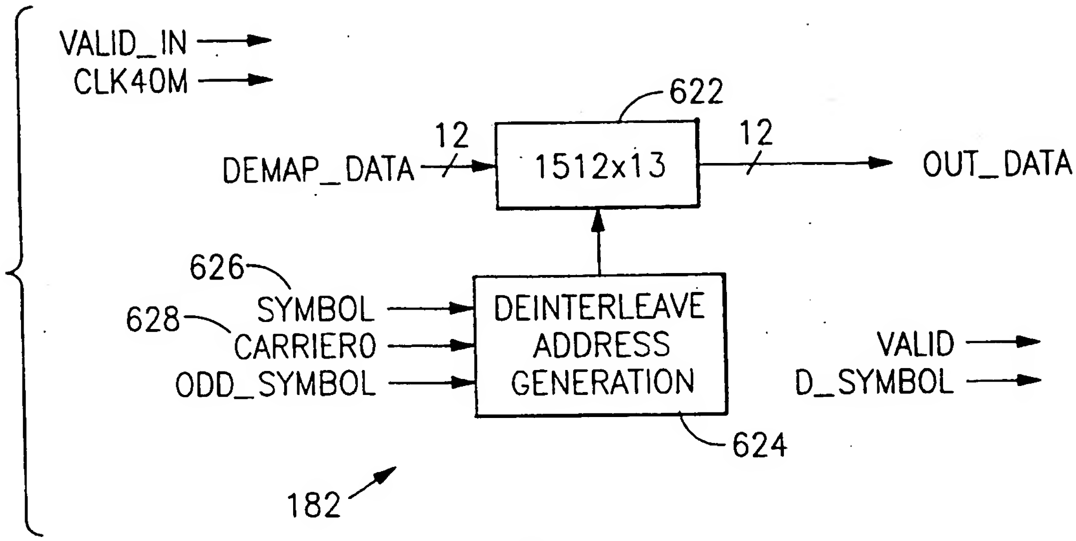
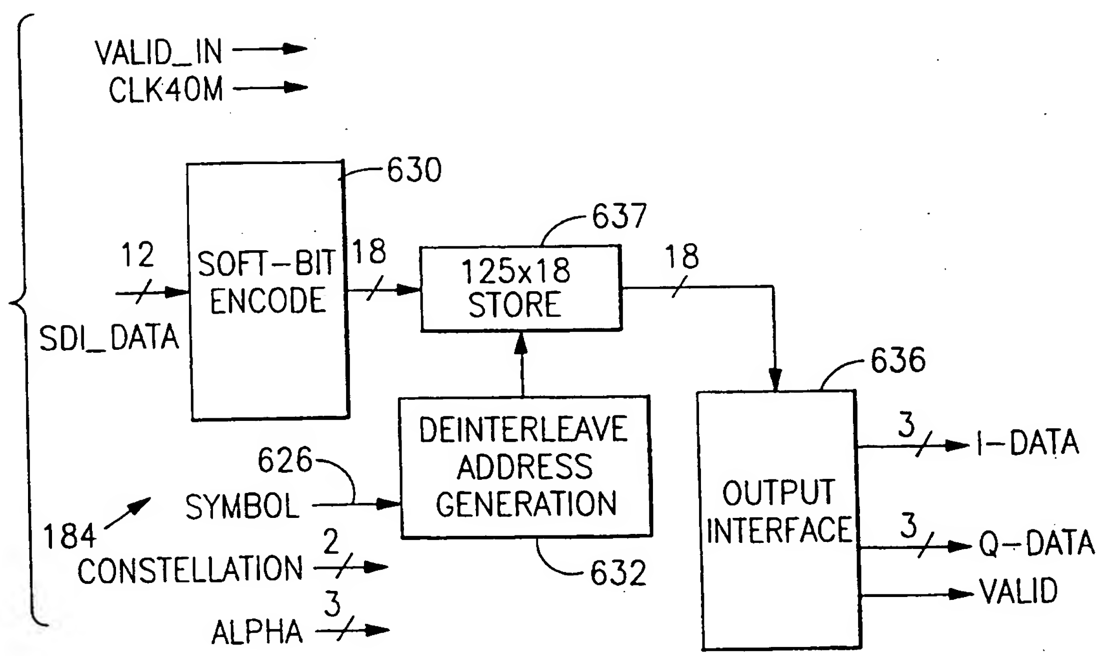

64-QAM
Q(1,3,5)
I(0,2,4)
BIT 5 SOFT
BIT 3 SOFT
BIT 4 SOFT
BIT 2 SOFT
I/Q DATA CONVERTED TO 3-BIT CONSTELLATION DATA VALUES AND SOFT-BITS
SOFT BIT CONVERSION LINEAR 0-7
000
001
011
010
110
111
101
100

FIG.51



VALID_IN
CLK40M
622
12
DEMAP_DATA
1512x13
12
OUT_DATA
626
SYMBOL
628
CARRIER0
ODD_SYMBOL
DEINTERLEAVE ADDRESS GENERATION
VALID
D_SYMBOL
624
182

FIG.52



VALID_IN
CLK40M
630
637
12
SOFT-BIT ENCODE
18
125x18 STORE
18
SDI_DATA
636
DEINTERLEAVE ADDRESS GENERATION
3
I-DATA
626
SYMBOL
OUTPUT INTERFACE
3
Q-DATA
184
2
CONSTELLATION
632
VALID
3
ALPHA

FIG.53

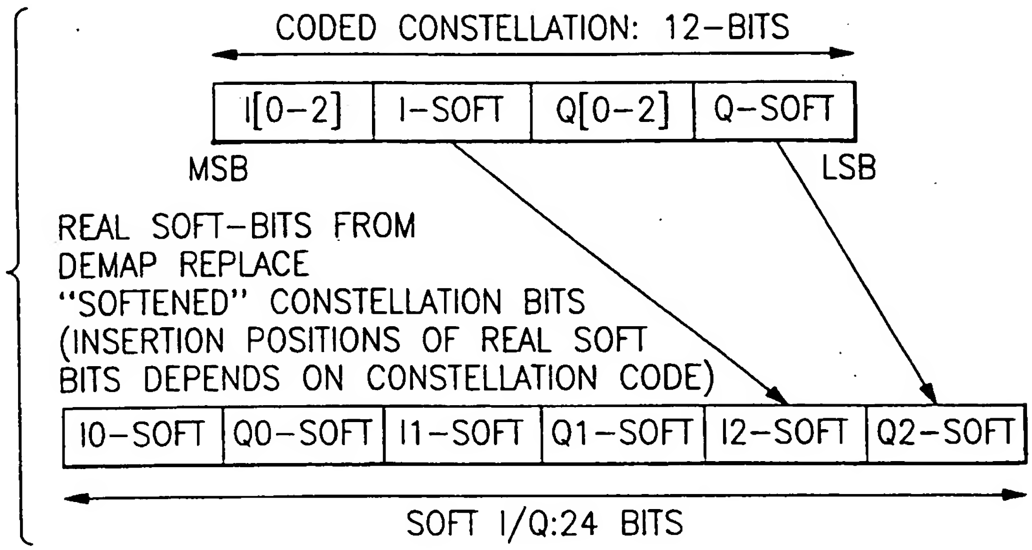

CODED CONSTELLATION: 12-BITS
I[0-2]
I-SOFT
Q[0-2]
Q-SOFT
MSB
LSB
REAL SOFT-BITS FROM DEMAP REPLACE "SOFTENED" CONSTELLATION BITS (INSERTION POSITIONS OF REAL SOFT BITS DEPENDS ON CONSTELLATION CODE)
I0-SOFT
Q0-SOFT
I1-SOFT
Q1-SOFT
I2-SOFT
Q2-SOFT
SOFT I/Q:24 BITS

FIG.54

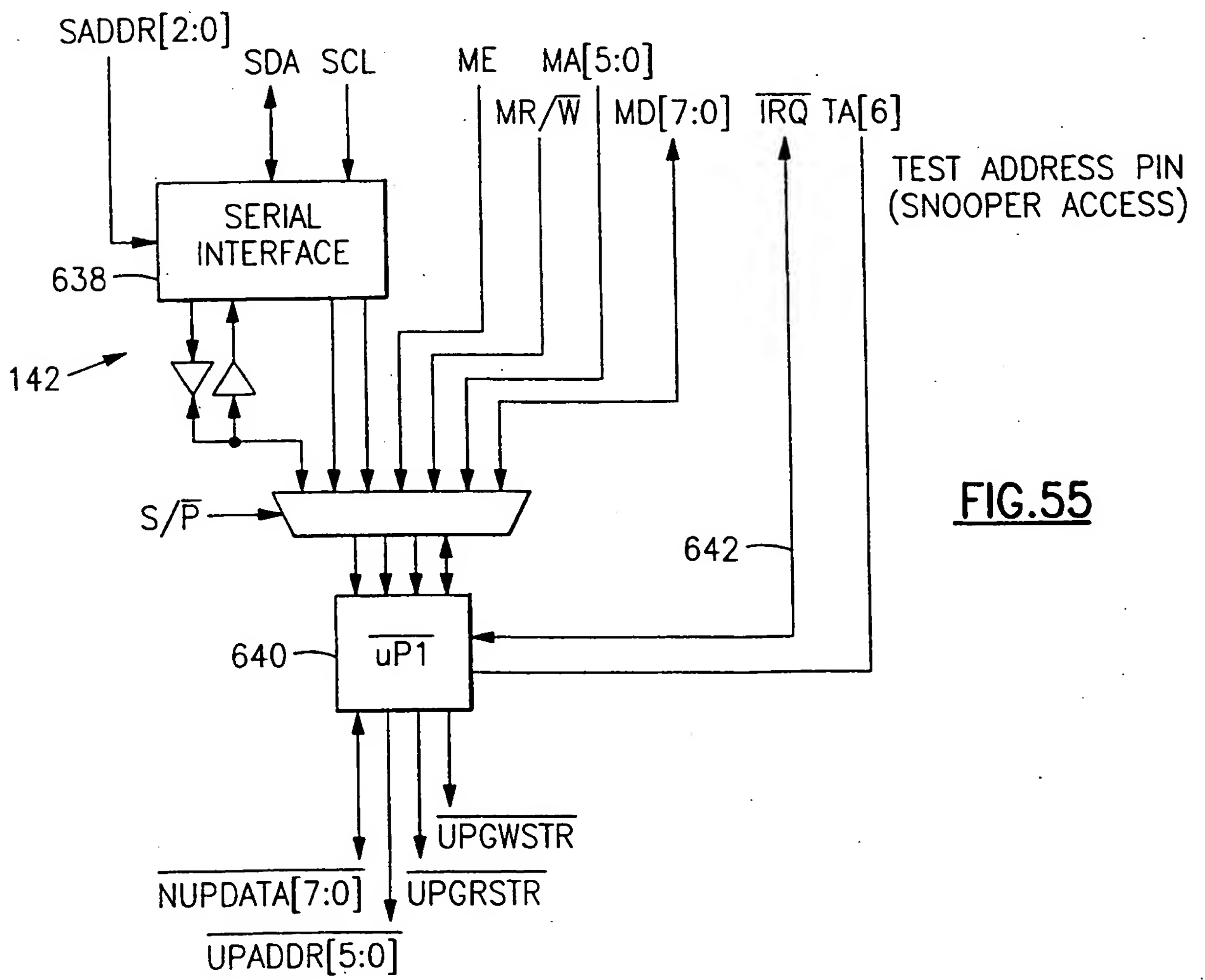

SADDR[2:0]
SDA
SCL
ME
MA[5:0]
MR/$\overline{W}$
MD[7:0]
$\overline{IRQ}$
TA[6]
TEST ADDRESS PIN (SNOOPER ACCESS)
SERIAL INTERFACE
638
142
S/$\overline{P}$
642
640
$\overline{uP1}$
$\overline{UPGWSTR}$
$\overline{NUPDATA[7:0]}$
$\overline{UPGRSTR}$
$\overline{UPADDR[5:0]}$

FIG.55

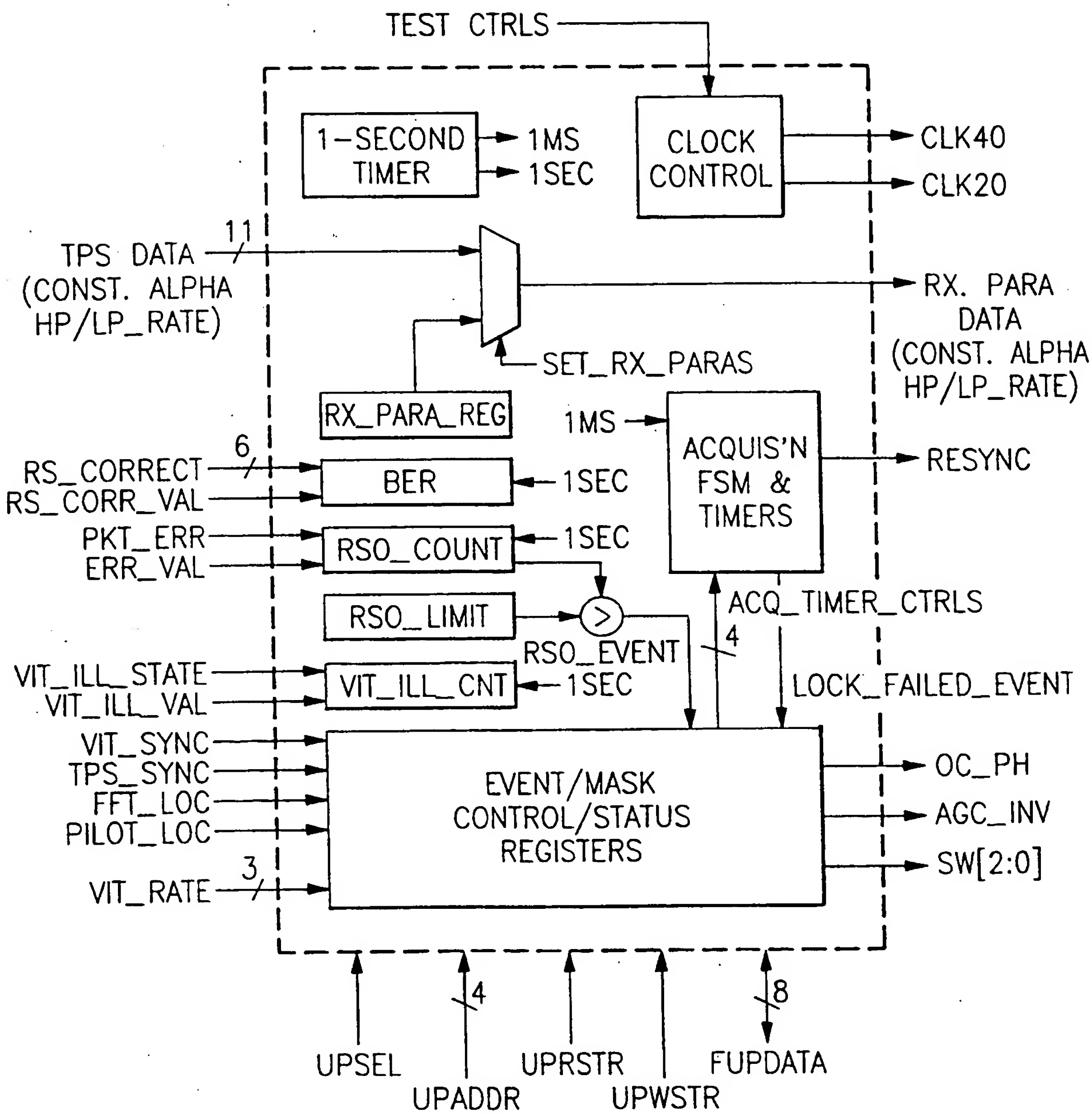

TEST CTRLS
1-SECOND TIMER
1MS
1SEC
CLOCK CONTROL
CLK40
CLK20
11
TPS DATA (CONST. ALPHA HP/LP_RATE)
RX. PARA DATA (CONST. ALPHA HP/LP_RATE)
SET_RX_PARAS
RX_PARA_REG
1MS
ACQUIS'N FSM & TIMERS
RESYNC
6
RS_CORRECT
RS_CORR_VAL
BER
1SEC
PKT_ERR
ERR_VAL
RSO_COUNT
1SEC
RSO_LIMIT
>
ACQ_TIMER_CTRLS
4
RSO_EVENT
LOCK_FAILED_EVENT
VIT_ILL_STATE
VIT_ILL_VAL
VIT_ILL_CNT
1SEC
VIT_SYNC
TPS_SYNC
FFT_LOC
PILOT_LOC
3
VIT_RATE
EVENT/MASK CONTROL/STATUS REGISTERS
OC_PH
AGC_INV
SW[2:0]
4
8
UPSEL
UPADDR
UPRSTR
UPWSTR
FUPDATA

FIG.56

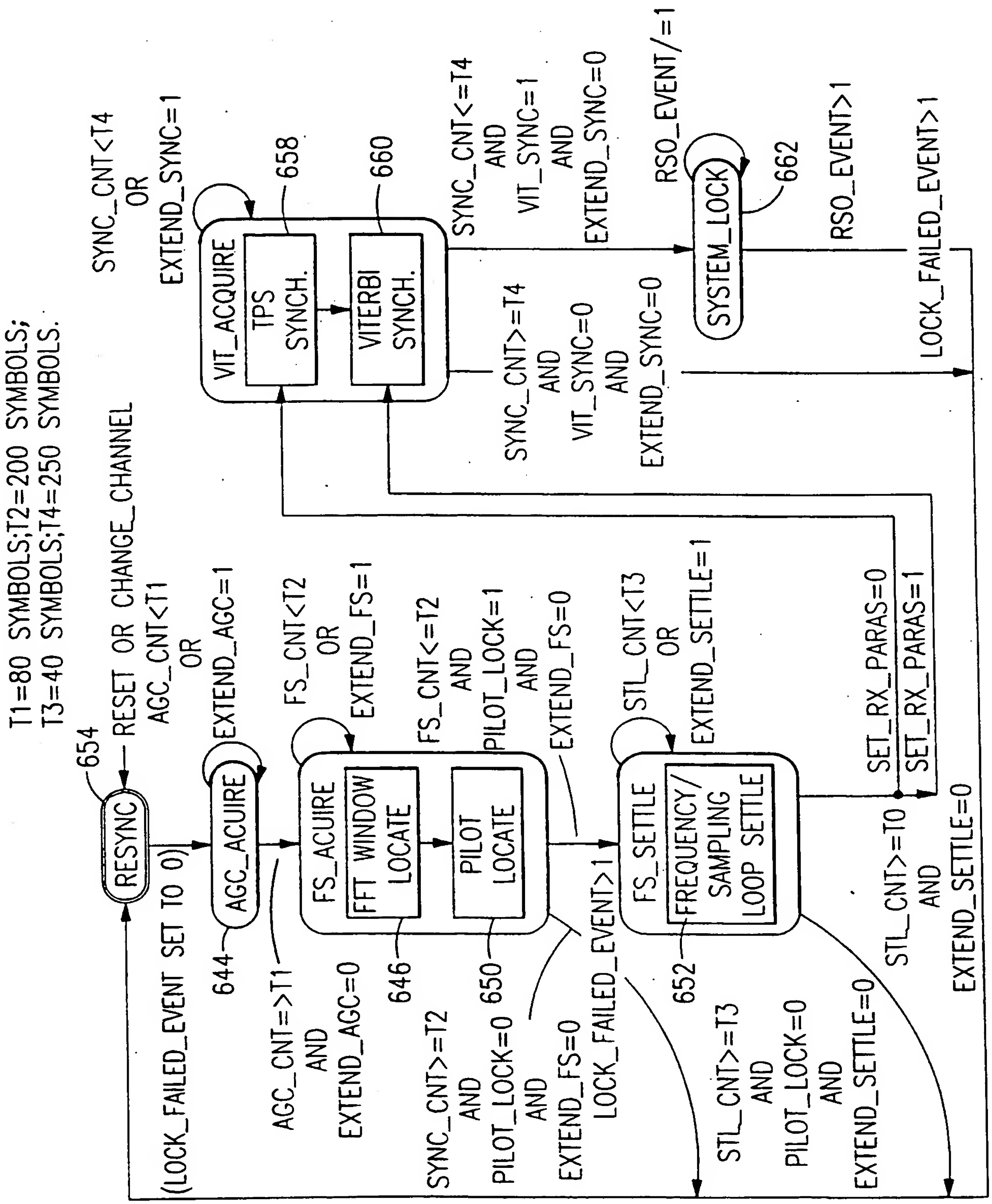

T1=80 SYMBOLS;T2=200 SYMBOLS;
T3=40 SYMBOLS;T4=250 SYMBOLS.
RESET OR CHANGE_CHANNEL
(LOCK_FAILED_EVENT SET TO 0)
654
RESYNC
AGC_CNT<T1
OR
EXTEND_AGC=1
AGC_ACUIRE
644
AGC_CNT=>T1
AND
EXTEND_AGC=0
FS_CNT<T2
OR
EXTEND_FS=1
FS_ACUIRE
FFT WINDOW
LOCATE
PILOT
LOCATE
646
650
SYNC_CNT>=T2
AND
PILOT_LOCK=0
AND
EXTEND_FS=0
LOCK_FAILED_EVENT>1
FS_CNT<=T2
AND
PILOT_LOCK=1
AND
EXTEND_FS=0
STL_CNT<T3
OR
EXTEND_SETTLE=1
FS_SETTLE
FREQUENCY/
SAMPLING
LOOP SETTLE
652
STL_CNT>=T3
AND
PILOT_LOCK=0
AND
EXTEND_SETTLE=0
SET_RX_PARAS=0
SET_RX_PARAS=1
STL_CNT>=T0
AND
EXTEND_SETTLE=0
SYNC_CNT<T4
OR
EXTEND_SYNC=1
VIT_ACQUIRE
TPS
SYNCH.
VITERBI
SYNCH.
658
660
SYNC_CNT>=T4
AND
VIT_SYNC=0
AND
EXTEND_SYNC=0
SYNC_CNT<=T4
AND
VIT_SYNC=1
AND
EXTEND_SYNC=0
RSO_EVENT/=1
SYSTEM_LOCK
662
RSO_EVENT>1
LOCK_FAILED_EVENT>1

FIG.57

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 27/26, G06F 17/14, H04L 1/00 — A3

(11) International Publication Number: WO 98/19410

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/18911

(22) International Filing Date: 22 October 1997 (22.10.97)

(30) Priority Data:

9622728.5	31 October 1996 (31.10.96)	GB
9720550.4	26 September 1997 (26.09.97)	GB

(71) Applicant: DISCOVISION ASSOCIATES [US/US]; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

(72) Inventors: ALAM, Dawood; 15 Westbury Park, Durdham Down, Bristol BS6 7JA (GB). COLLINS, Matthew, James; Flat 1, 4 New King Street, Bath BA1 2BN (GB). DAVIES, David, Huw; 6 Glen Brook, Glen Drive, Stoke Bishop, Bristol BS9 1SB (GB). KEEVILL, Peter, Anthony; 7 Junction Road, Oldfield Park, Bath BA2 3NQ (GB). NOLAN, John, Matthew; 19 The Firs, Combe Down, Bath, Somerset BA2 5ED (GB). FOXCROFT, Thomas; 52B Pembroke Road, Clifton, Bristol BS8 3DT (GB). PARKER, Jonathan; 66 Third Avenue, Oldfield Park, Bath BA2 3NZ (GB).

(74) Agent: BICKEL, Arthur, S.; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

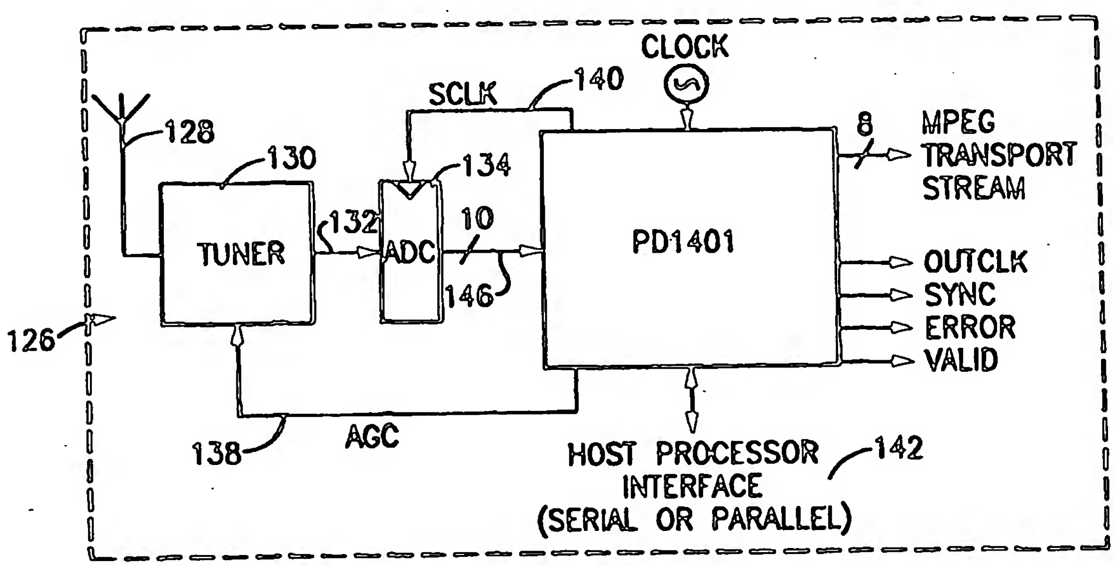
(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report: 27 August 1998 (27.08.98)

(54) Title: SINGLE CHIP VLSI IMPLEMENTATION OF A DIGITAL RECEIVER EMPLOYING ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING



(57) Abstract

The invention provides a single chip implementation of a digital receiver for multicarrier signals that are transmitted by orthogonal frequency division multiplexing. Improved channel estimation and correction circuitry are provided. The receiver has highly accurate sampling rate control and frequecy control circuitry. BCH decoding of tps data carriers is achieved with minimal resources with an arrangement that includes a small Galois field multiplier. An improved FFT window synchronization circuit is coupled to the resampling circuit for locating the boundary of the guard interval transmitted with the active frame of the signal. A real-time pipelined FFT processor is operationally associated with the FFT window synchronization circuit and operates with reduced memory requirements.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 97/18911

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L27/26 G06F17/14 H04L1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 730 357 A (TELIA AB) 4 September 1996 see abstract see figure 1 see figure 10	1,4
A		2

Y	US 4 300 229 A (HIROSAKI BOTARO) 10 November 1981 see abstract see column 15, line 15 - line 32; figure 4 see column 39, line 59 - line 60	1,4
A		

A	EP 0 653 858 A (TOKYO SHIBAURA ELECTRIC CO) 17 May 1995 see abstract see figure 5	1,2

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

2

Date of the actual completion of the international search	Date of mailing of the international search report
5 March 1998	07.07.98

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Koukourlis, S

INTERNATIONAL SEARCH REPORT

Inte[rnati]onal Application No
PCT/US 97/18911

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WIRELESS PERSONAL COMMUNICATIONS, vol. 2, no. 4, 1 January 1996, pages 321-334, XP000589621 WYATT-MILLINGTON W ET AL: "A PIPELINED IMPLEMENTATION OF THE WINOGRAD FFT FOR SATELLITE ON-BOARD MULTI-CARRIER DEMODULATION" see page 321, paragraph 2 - paragraph 3	1
A	WO 95 03656 A (TELIA AB ;ISAKSSON MIKAEL (SE); ENGSTROEM BO (SE)) 2 February 1995 see abstract see page 7, line 15 - line 20 see figure 4	1
A	WO 96 24989 A (ADC TELECOMMUNICATIONS INC) 15 August 1996 see page 61, line 26 - page 62, line 6 see page 84, line 4 - page 85, line 9 see figures 11,27	1
A	EP 0 722 235 A (MATSUSHITA ELECTRIC IND CO LTD) 17 July 1996 see figure 2	1
A	EP 0 689 314 A (NOKIA TECHNOLOGY GMBH) 27 December 1995 see figure 2	1

2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 97/ 18911

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1-4: Receiver for multicarrier signals comprising FFT synchronisation circuit for locating a boundary of the guard interval;
5-7,10-24: functions performed by the FFT processor;
8,9,27-35: channel estimation and correction
25,26: BCH decoder

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1 - 4

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 97/18911

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0730357 A	04-09-96	NO 960759 A	02-09-96
		SE 9500743 A	02-09-96
US 4300229 A	10-11-81	JP 1496399 C	16-05-89
		JP 55112054 A	29-08-80
		JP 63046619 B	16-09-88
		JP 1496400 C	16-05-89
		JP 55112055 A	29-08-80
		JP 63046620 B	16-09-88
		JP 1496401 C	16-05-89
		JP 55112056 A	29-08-80
		JP 63046621 B	16-09-88
		JP 1478950 C	10-02-89
		JP 56093449 A	29-07-81
		JP 63026575 B	30-05-88
		AU 527333 B	24-02-83
		AU 5563480 A	28-08-80
		CA 1134519 A	26-10-82
EP 0653858 A	17-05-95	JP 7143097 A	02-06-95
		CA 2135970 A	17-05-95
		US 5602835 A	11-02-97
WO 9503656 A	02-02-95	SE 500986 C	17-10-94
		EP 0712555 A	22-05-96
		SE 9302453 A	17-10-94
		US 5652772 A	29-07-97
WO 9624989 A	15-08-96	AU 4915896 A	27-08-96
		AU 4916296 A	27-08-96
		BR 9607031 A	04-11-97
		CA 2211117 A	15-08-96
		CA 2211803 A	15-08-96
		CZ 9702487 A	17-12-97
		EP 0809898 A	03-12-97
		EP 0808534 A	26-11-97
		FI 973134 A	06-10-97
		WO 9624995 A	15-08-96
EP 0722235 A	17-07-96	CA 2166599 A	11-07-96

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 97/18911

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0722235 A		JP 8251135 A	27-09-96
EP 0689314 A	27-12-95	FI 942876 A	17-12-95